

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой №43

должность, уч. степень, звание	подпись, дата	инициалы, фамилия
--------------------------------	---------------	-------------------

БАКАЛАВРСКАЯ РАБОТА

на тему Разработка удаленного анализатора USB-трафика

выполнена Солтановым Ильей Назимовичем

фамилия, имя, отчество студента в творительном падеже

по направлению подготовки 09.03.04 «Программная инженерия»

код

наименование направления

направленности 01 «Разработка программно -
информационных систем»

наименование направления

код

наименование направленности

Студент группы № 4332 И.Н. Солтанов

подпись, дата

инициалы, фамилия

Руководитель

доц., к.т.н., доц.

А.А. Ключарев

должность, уч. степень, звание

подпись, дата

инициалы, фамилия

Консультанты

- по конструкционно-технологическим вопросам

асс.

К.В. Бабанов

должность, уч. степень, звание

подпись, дата

инициалы, фамилия

Санкт-Петербург 2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой №43

должность, уч. степень, звание

подпись, дата

инициалы, фамилия

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ БАКАЛАВРСКОЙ РАБОТЫ

студенту группы

4332

номер

Солтанову Илье Назимовичу

фамилия, имя, отчество

на тему

Разработка удаленного анализатора USB-трафика

утвержденную приказом ГУАП от

11.04.17

№

01-301

Цель работы:

Разработать программно-аппаратное устройство, позволяющее

отслеживать и модифицировать поток данных по каналу USB, в целях мониторинга

коррекции ошибок, отладки и исследуемой системой

Задачи, подлежащие решению:

Извлечение USB-трафика, обработка полученной

информации, хранение результатов обработки, управление исследуемой системы,

управление анализатором трафика, организация клиент-серверного взаимодействия.

Содержание работы (основные разделы):

Описание предметной области,

проектирование, сетевое взаимодействие, разработка аппаратной части,

разработка программной части, результаты разработки, руководство пользователя.

Срок сдачи работы « ____ »

201 ____

Руководитель

доц., к.т.н., доц.

должность, уч. степень, звание

подпись, дата

А.А. Ключарев

инициалы, фамилия

Задание принял(а) к исполнению

студент группы №

4332

подпись, дата

И.Н. Солтанов

инициалы, фамилия

РЕФЕРАТ

Выпускная квалификационная работа содержит 153 с., 32 рис., 2 табл., 20 источников, 8 прил.

АНАЛИЗ ТРАФИКА USB, ЯЗЫК ПРОГРАММИРОВАНИЯ C++, БД, СУБД, MYSQL, USB, АРХИТЕКТУРА «КЛИЕНТ-СЕРВЕР», ТРЕХУРОВНЕВАЯ АРХИТЕКТУРНАЯ МОДЕЛЬ, СЕРВЕР, КЛИЕНТ, USB HID, КЛАВИАТУРА, АППАРАТНАЯ ПЛАТФОРМА ARDUINO, GSM, GPRS, GSM/GPRS-МОДУЛЬ, ПЛАТЫ РАСШИРЕНИЯ ARDUINO, ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ.

Объектом исследования и разработки является устройство, осуществляющее анализ USB трафика.

Цель работы - осуществление возможности без вмешательства в исследуемую систему отслеживать поток данных по USB каналу связи, и при необходимости изменять его, в целях исправления ошибок, отладки, или управления исследуемой системой.

В процессе работы использовались программные средства:

- StarUML.
- JetBrains CLion 2016.3.2 x64.
- JetBrains DataGrip 2017.1.3 x64.
- Fritzing.
- PlatformIO IDE 3.2.1.

Полученные результаты и их новизна:

В результате была разработано устройство, которое осуществляет анализ трафика USB, проходящего между ведущим и ведомым устройством. Для взаимодействия с полученными данными, для контроля устройства разработана трехуровневая архитектура «клиент-сервер», состоящая из сервера базы данных, сервера приложений, пользовательского клиента.

В качестве ведомого устройства выступает клавиатура, а в качестве ведущего – персональный компьютер.

Экономическая эффективность или значимость работы:

Система позволит упростить ведение аудита для вычислительных устройств, не имеющих операционной системы с графической оболочкой, или не имеющих выход в сеть Интернет, а также позволит упростить процессы тестирования и отладки программного обеспечения, разрабатываемого для USB-устройств.

В будущем планируется модернизировать проект в коммерческих целях.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	10
1 Описание предметной области	13
1.1 Цели и задачи.....	13
1.1.1 Назначение системы и постановка задач	13
1.1.2 Актуальность работы	14
1.2 Характеристика решаемых задач.....	14
1.2.1 Сбор и обработка информации	14
1.2.2 Удаленное управление исследуемой системы	15
1.3 Известные аналоги системы.....	15
1.4 Практическая ценность работы.....	16
2 Проектирование.....	17
2.1 Анализ трафика.....	17
2.2 Сетевое взаимодействие.....	21
2.2.1 Архитектура «клиент-сервер»	21
2.2.2 Взаимодействие средствами сотовой связи	24
2.3 Разработка аппаратной части	25
2.3.1 Анализ аналогов аппаратных платформ	26
2.3.1.1 Платформа STM32VLDDiscovery.....	26
2.3.1.2 Платформа Raspberry Pi (model B).....	27
2.3.1.3 Платформа Arduino Leonardo	29
2.3.1.4 Сравнительная характеристика аппаратных платформ....	30
2.3.2 Анализ аналогов модулей сотовой связи	32
2.3.2.1 GSM/GPRS модуль SIM800L	33

2.3.2.2	GSM/GPRS модуль Ai-Thinker A6	34
2.3.2.3	Сравнительная характеристика GSM/GPRS модулей	34
2.3.3	Плата расширения USB Host Shield 2.0	36
2.4	Схема подключения	38
2.5	Разработка программной части	41
2.5.1	Серверная часть	42
2.5.1.1	Сервер базы данных	42
2.5.2.1	Сервер приложений	45
2.5.2.1.1	Описание модулей сервера приложений	48
2.5.2.1.1.1	Менеджер соединений	49
2.5.2.1.1.2	Основной модуль сервера	49
2.5.2.1.1.3	Модуль взаимодействий	50
2.5.2.1.1.4	Модуль базы данных	51
2.5.2.1.1.5	Модуль обработчика команд	53
2.5.2.1.1.6	Модуль настроек	56
2.5.3	Клиентская часть	57
2.5.3.1	Описание модулей сервера приложений	59
2.5.3.1.1	Менеджер соединений	59
2.5.3.1.2	Модуль взаимодействий	60
2.5.3.1.3	Основной модуль	60
2.5.3.1.4	Модуль настроек	61
2.5.4	Программное обеспечение анализатора трафика	61
2.5.4.1	Описание модулей сервера приложений	64
2.5.4.1.1	Основной модуль	65

2.5.4.1.2	Модуль связи.....	66
2.5.4.1.3	Модуль обработчика клавиатуры	68
2.5.4.1.5	Звуковой модуль	69
2.5.4.1.6	Модуль анализа трафика	70
2.5.5	Проектирование корпуса устройства	71
3	Результаты разработки	74
3.1	Аппаратная часть	74
3.2	Программная часть	75
4	Руководство пользователя.....	76
4.1	Область применения.....	76
4.2	Требования к уровню подготовки.....	76
4.3	Требования к программной среде.....	76
4.4	Настройка сервера приложений.....	77
4.5	Настройка пользовательского клиента.....	77
4.6	Настройка анализатора трафика	77
4.8	Описание операций сервера приложений	79
4.9	Описание пользовательских операций анализатора трафика	80
4.10	Описание операций пользовательского клиента	81
5	Заключение	84
	СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	87
	ПРИЛОЖЕНИЕ А.....	89
	ПРИЛОЖЕНИЕ Б.....	102
	ПРИЛОЖЕНИЕ В	110
	ПРИЛОЖЕНИЕ Г.....	147

ПРИЛОЖЕНИЕ Д	149
ПРИЛОЖЕНИЕ Е	151
ПРИЛОЖЕНИЕ Ж	152
ПРИЛОЖЕНИЕ З	153

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

БД – база данных.

СУБД – система управления базами данных.

GSM – Global System for Mobile Communications – глобальная система мобильной связи.

GPRS – General Packet Radio Service — пакетная радиосвязь общего пользования.

SQL – Structured Query Language — язык структурированных запросов.

HID – Human Interface Device – устройство интерфейса пользователя.

USB – Universal Serial Bus — универсальная последовательная шина.

ВВЕДЕНИЕ

Тема анализа трафика является достаточно обширной. Под «анализом трафика» понимается набор технологий и их реализаций, которые позволяют осуществлять сбор, обработку, контроль и изменение пакетов, проходящих по анализируемому интерфейсу в реальном времени, в зависимости от содержимого.

Существует множество проблем, связанных с анализом трафика на расстоянии. Эти проблемы связаны с областью применения анализа, будь то исследование трафика отдельно взятого модуля системы, который не имеет операционной системы, или же самой системы, чей трафик просматривается, обрабатывается и подвергается управляемому изменению (осуществление управления исследуемой системы).

Основной проблемой анализа трафика USB на расстоянии является проведение исследования объекта (системы или конкретного устройства) не затрагивая работу и взаимодействие ведущего и ведомого устройств. Для осуществления данной цели производится разработка аппаратного обеспечения, которое будет находиться между ведущим и ведомым устройствами и не будет производить какие-либо затраты (как временные, так и ресурсные), влияющие на их взаимодействие. Интерес к проблеме незаметной для пользователя передачи трафика указывает на то, что решение данной задачи актуально, и необходимо найти способ её решения, который оказывал бы минимальное влияние на работоспособность канала связи USB и на взаимодействие устройств, осуществляющих передачу данных по каналу связи между собой через анализатор трафика.

Цель данной выпускной квалификационной работы - решение вышеупомянутой проблемы, которая заключается в наблюдении за потоком данных, проходящим по тому или иному каналу связи между ведущим и ведомым USB-устройствами на аппаратном уровне, а также реализации

возможности при необходимости производить умышленное изменение трафика, в целях исправления ошибок, проведения отладки, или управления исследуемой системой. В качестве ведомого устройства будет выступать клавиатура, а в качестве ведущего – персональный компьютер.

Благодаря этому достигается возможность проведения исследования системы без какого-либо неумышленного вмешательства непосредственно в ее работу с ведомым устройством, что также позволяет значительно расширить круг анализируемых систем.

Для достижения этой цели предполагается выполнение следующих шагов:

- Разработка устройства, осуществляющего сбор, обработку, а также хранение информации на удаленном сервере, и осуществляющее её извлечение из USB трафика, проходящего по каналу связи между ведущим и ведомым USB-устройствами.
- Разработка клиент-серверного приложения, главными задачами которого являются:

Со стороны сервера:

- возможность работы с множеством клиентов одновременно;
- хранение и обработка полученной информации;
- управление клиентами.

Со стороны клиента:

- предварительный сбор и обработка USB-трафика;
- отправка обработанных данных на сервер;
- обработка команд сервера.

Процесс сбора информации производится путем захвата трафика, проходящего через USB-соединения исследуемой системы. Полученные данные отправляются на удаленный сервер, где хранятся для дальнейшей обработки. Преимущество в использовании удаленного сервера заключается в том, что

сервер позволяет удаленно просматривать и выполнять анализ передаваемых данных по интерфейсу USB.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Цели и задачи

1.1.1 Назначение системы и постановка задач

Основной целью данной выпускной квалификационной работы является осуществление возможности без вмешательства в исследуемую систему отслеживать поток данных по USB каналу связи, и при необходимости изменять его, в целях исправления ошибок, отладки, или управления исследуемой системой.

В качестве ведомого устройства будет выступать клавиатура, а в качестве ведущего – персональный компьютер.

Основными задачами данной выпускной квалификационной работы являются:

- 1) Проведение обработки, а также хранение информации на удаленном сервере.
- 2) Осуществление извлечения из USB трафика, проходящего по каналу связи между ведущим и ведомым USB-устройствами.
- 3) Разработка клиент-серверного приложения, главными задачами которого являются:

Со стороны сервера:

- возможность работы с множеством клиентов одновременно;
- хранение и обработка полученной информации;
- управление клиентами.

Со стороны клиента:

- предварительный сбор и обработка USB-трафика;
- отправка обработанных данных на сервер;
- обработка команд сервера.

1.1.2 Актуальность работы

Среди проблем, связанных с анализом трафика на расстоянии, пристальное внимание уделяется способам передачи информации, полученной с USB трафика без вмешательства в работу ведущего и ведомого устройств.

Интерес к проблеме незаметной для пользователя передачи трафика указывает на то, что её решение является актуальным.

Разработка решения для данной проблемы позволит упростить ведение аудита для вычислительных устройств, не имеющих операционной системы с графической оболочкой, или не имеющих выход в сеть Интернет, а также позволит упростить процессы тестирования и отладки программного обеспечения, разрабатываемого для USB-устройств.

Также следует упомянуть, что на текущий момент на рынке отсутствуют дешевые аналоги разрабатываемого устройства, которые обладают функциональными возможностями анализа потока данных по USB каналу связи без вмешательства в работу исследуемой системы.

1.2 Характеристика решаемых задач

1.2.1 Сбор и обработка информации

Анализ USB трафика заключается в наблюдении за потоком данных, проходящим по тому или иному каналу связи между ведущим и ведомым USB-устройствами на аппаратном уровне, т. е. анализ системы происходит без какого-либо вмешательства в работу системы, что позволяет значительно расширить круг анализируемых систем.

Процесс сбора информации производится путем захвата трафика, проходящего через USB-соединения исследуемой системы. Захват осуществляется обработчиком поступивших пакетов данных в канале связи. Полученные данные из двоичного формата декодируются в удобный для человека вид (набор шестнадцатеричных кодов, набор ASCII-кодов) и

отправляются на удаленный сервер, позволяя удаленно просматривать и выполнять анализ передаваемых пакетов данных. Стоит отметить, что после обработки пакеты данных отправляются истинному получателю.

1.2.2 Удаленное управление исследуемой системы

Имея возможность перехватывать пакеты данных, можно осуществлять удаленный контроль путем отправки пакетов, где в роли получателя будет выступать исследуемая система. Эта особенность позволяет также осуществлять корректировку передаваемых данных в целях исправления ошибок, а также для проведения отладки исследуемой системы. Контроль производится удаленно, с использованием сервера, или службы коротких сообщений мобильной связи.

1.3 Известные аналоги системы

Стоит выделить следующие продукты, задачей которых является анализ USB-трафика:

- Free USB Analyzer - бесплатное программное обеспечение, разработанное компанией HND Software Ltd. Задача данного программного обеспечения заключается в анализе и мониторинге USB-устройств. Данный программный продукт используется для захвата, перехвата и декодирования полученного трафика, который проходит через USB-интерфейсы к подключенным рабочей станции, устройствам;
- Beagle – коммерческий продукт, разработанный компанией Total Phase. Данный продукт представляет собой устройство, целью которого является перехват трафика между USB устройством и компьютером, запись полученной информации производится на носитель с flash-памятью.

Преимущество данной выпускной квалификационной работы перед представленными аналогами заключается в том, что:

- для работы устройства не нужна установка программного обеспечения на исследуемые системы;

- полученная информация отправляется на удаленный сервер.

Ключевое отличие – для передачи данных используется GSM-модуль, и, соответственно, для работы модуля необходимо наличие SIM-карты.

1.4 Практическая ценность работы

Практическая ценность данной работы заключается в достижении следующих результатов:

- 1) Удобство использования устройства позволяет упростить управление USB-трафиком
- 2) Незаметный перехват и управление USB-трафиком при тестировании и отладке USB-устройства позволяет снизить нагрузку на исследуемую систему (возможность вести тестирование и отладку без взаимодействия с ведущим устройством).
- 3) Использование GSM/GPRS-модуля позволяет осуществлять передачу данных вне зависимости от наличия выхода в сеть Интернет исследуемой системы.
- 4) Удаленный доступ к хранимым на сервере данным, полученным в результате работы устройства увеличивает мобильность работы проекта, путем подключения к серверу используя клиентское приложение.
- 5) Повышение эффективности взаимодействия исследователя с хранимыми данными на сервере за счет фиксирования времени внесения и изменения новых записей в базе данных, фиксирование прочтенных и удаленных данных позволяет минимизировать затраты на процессы обработки и анализа USB-трафика.

2 ПРОЕКТИРОВАНИЕ

2.1 Анализ трафика

Анализ трафика исследуемой системы заключается в наблюдении за потоком данных, проходящему по каналу связи между ведущим и ведомым устройствами на аппаратном уровне. Процесс сбора информации производится путем захвата трафика, проходящего через соединения исследуемой системы.

Трафик проходит по каналу связи USB. Блок данных, передаваемый между устройствами, называется USB-кадром или USB-фреймом. Кадр передается за точно определенный интервал времени. Взаимодействие с блоками данных и командами осуществляется при помощи абстракции, именуемой каналом. Стоит отметить, что внешнее устройство называют конечной точкой, поскольку оно также представляет собой некую абстракцию. Таким образом, канал связи представляет собой логическую связку между устройствами [1].

Для организации пересылки команд и тех данных, которые входят в состав этих команд, применяются каналы связи по умолчанию, а для осуществления пересылки данных используются потоковые каналы, или же каналы сообщений [2].

По каналу связи информация передается в виде блока данных именуемых пакетами (Packet). Пакет данных имеет следующую структуру:

- поле SYNC (SYNChronization) синхронизации;
- поле PID (Packet IDentifier) идентификатора пакета;
- поле Check, которое является побитовой инверсией идентификатора пакета;
- поле данных пакета;
- поле EOP (End of Packet) - последовательность бит определяющее конец пакета.

Схема структуры пакета данных представлена на Рис.1.

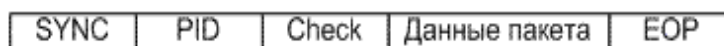


Рисунок 1 – Структура пакета USB

Обмен данными происходит при помощи транзакций. Транзакция представляет собой неразрывные последовательности пакетов. Обмен данными начинается с ведущего устройства, которое становится инициатором обмена. Инициатор начинает передачу с короткого пакета, который сообщает о начале новой транзакции. Данный пакет именуется как токен-пакет (token). В этом пакете ведущий определяет номер точки окончания (endpoint), адрес ведомого устройства, а также направление транзакции (IN или OUT). Токен направления транзакции OUT сообщает о том, что незамедлительно после него прибудет пакет с данными. В транзакции может быть несколько пакетов с данными при условии, что размер каждого из этих пакетов составляет максимально допустимый для устройства объем данных [3]. Завершение передачи данных идентифицируется по пакету, размер которого не является максимальным. В момент, когда приходит пакет меньшей длины, устройство незамедлительно передаёт пакет (handshake), который является ответным пакетом для подтверждения. К примеру, всё успешно принято - ACK, не все данные были приняты - NACK, данные назначены отключенному номеру точки окончания - STALL. В транзакции пакеты передаются практически вплотную, поскольку максимальная между пакетами задержка не должна быть больше ~1 мкс [4]. Если данное условие не выполняется, то транзакция будет определена как ошибочная. Стоит учитывать то, что если передача данных осуществляется транзакциями, то следует иметь в виду, что изменение паузы повлечёт отмену транзакции.

USB представляет собой сеть, состоящую из с одного ведущего (мастера) и некотором количеством ведомых устройств (подчинённые устройства). Топология сети представляет собой активное дерево. Это означает, то, что в каждом узле этого дерева расположено специальное устройство, именуемое как хаб, или же концентратор.

Задача концентратора заключается в маршрутизации пакетов, в электрическом согласовании кабелей, в определении подключения и отключения устройств. А так как для того, чтобы избежать проблем, связанных с передачей кадров, конечное устройство выступает в роли USB-хаба. Схема передачи пакетов представлена на Рис. 2.



Рисунок 2 – Схема передачи кадров между ведущим, ведомым устройствами используя анализатор трафика

Анализатор USB-трафика выполняет работу концентратора, а именно осуществляет ретрансляцию входящих пакетов с одного USB-порта в другой. Для сбора трафика и дальнейшего анализа устройство не просто ретранслирует входящие кадры, но и копирует кадры, и производит их обработку. Обработка скопированных пакетов заключается в том, что из кадров извлекается информация поля данных пакета, а затем полученные данные из двоичного формата декодируются в удобный для человека вид (набор шестнадцатеричных кодов, набор ASCII-кодов) (См. Рис.3).

Результаты обработки кадров хранятся в буфере устройства. Как только размер буфера достигнет критической отметки, то незамедлительно осуществляется отправка данных на удаленный сервер. Передача данных осуществляется с использованием стека протоколов TCP/IP, вне зависимости от

наличия подключения к сети Интернет исследуемого объекта, трафик которого анализируется.

Хранение данных в буфере, позволяет осуществлять удаленный контроль путем отправки пакетов ведущему устройству с целью корректировки передаваемых данных (исправления ошибок передачи), а также для проведения отладки исследуемой системы. Как и передача хранимых данных, управление ведущим устройством производится удаленно (См. Рис.3).

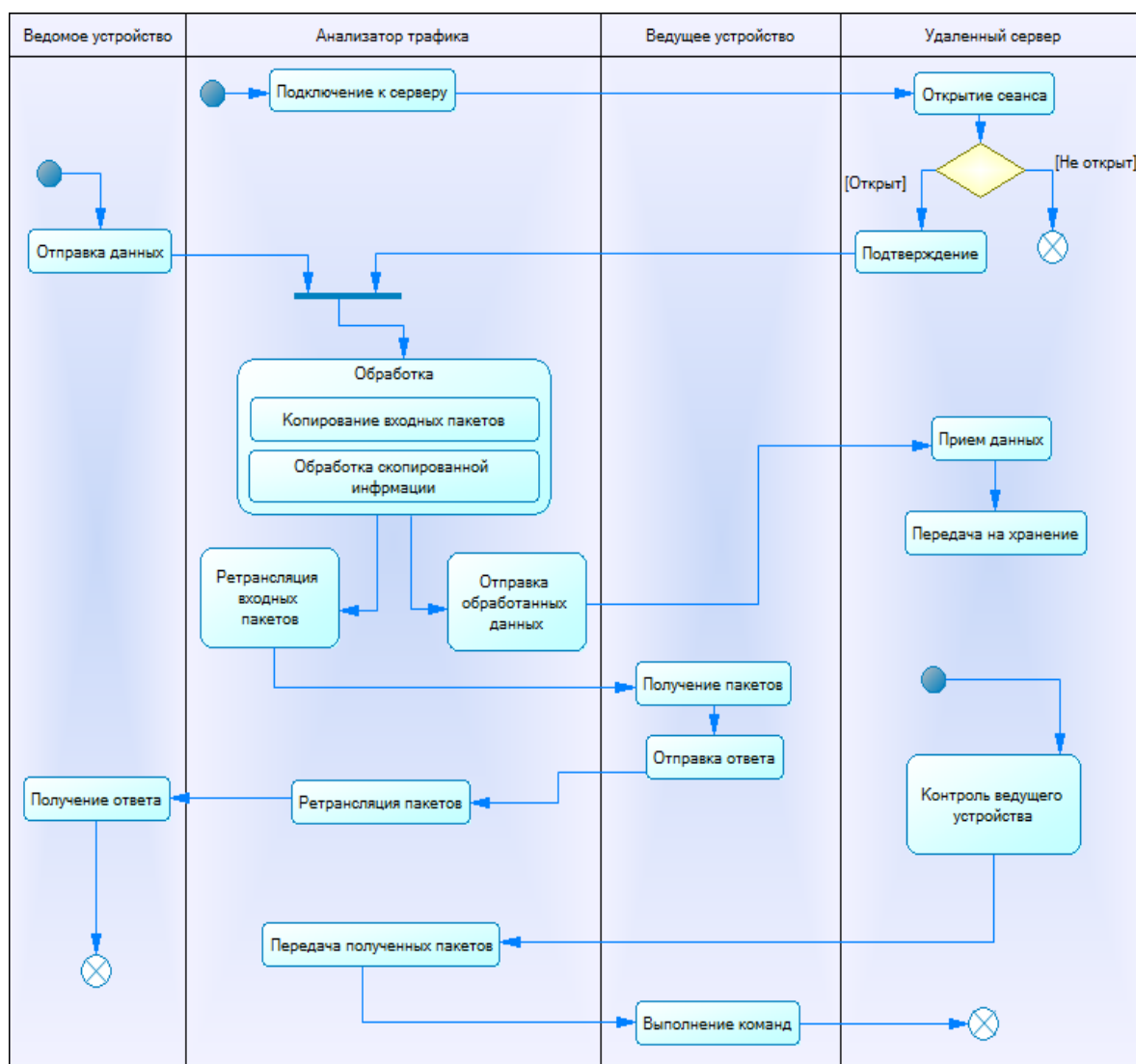


Рисунок 3 – Диаграмма деятельности, отражающая взаимодействие с анализатором трафика

2.2 Сетевое взаимодействие

Устройство для анализа трафика подключается к исследуемой системе. Наличие подключения к сети Интернет исследуемой системы не имеет значения для нее работа анализатора не заметна, т.к. устройство не вмешивается в работу системы. Для связи с сервером используется протокол TCP/IP. Подключение осуществляется при помощи сотовой связи по технологиям GSM/GPRS. Соответственно для корректной работы анализатора потребуется SIM-карта с возможностью выхода в сеть Интернет.

При подключении анализатора USB-трафика к исследуемой системе осуществляется его запуск. Производится инициализация обработчиков входных и выходных сигналов, осуществляется идентификация устройства как USB-hub. Заключительным этапом является подключение к серверу приложений, который регистрирует подключенное устройство для дальнейшей работы с ним.

Хранение данных на сервере позволяет удаленно просматривать и выполнять анализ передаваемых пакетов данных между ведущим и ведомыми устройствами по каналу связи USB. Для просмотра данных необходимо запросить необходимую информацию у сервера предварительно осуществив подключение к нему.

2.2.1 Архитектура «клиент-сервер»

Сетевая архитектура представляет из себя трёхуровневую архитектурную модель (Рис. 4), в которой присутствуют следующие слои [5]:

- слой клиента: клиент, анализатор трафика;
- слой логики: сервер приложений (сервер, к которому подключается клиент);
- слой данных: сервер базы данных (сервер, с которым работает сервер приложений).

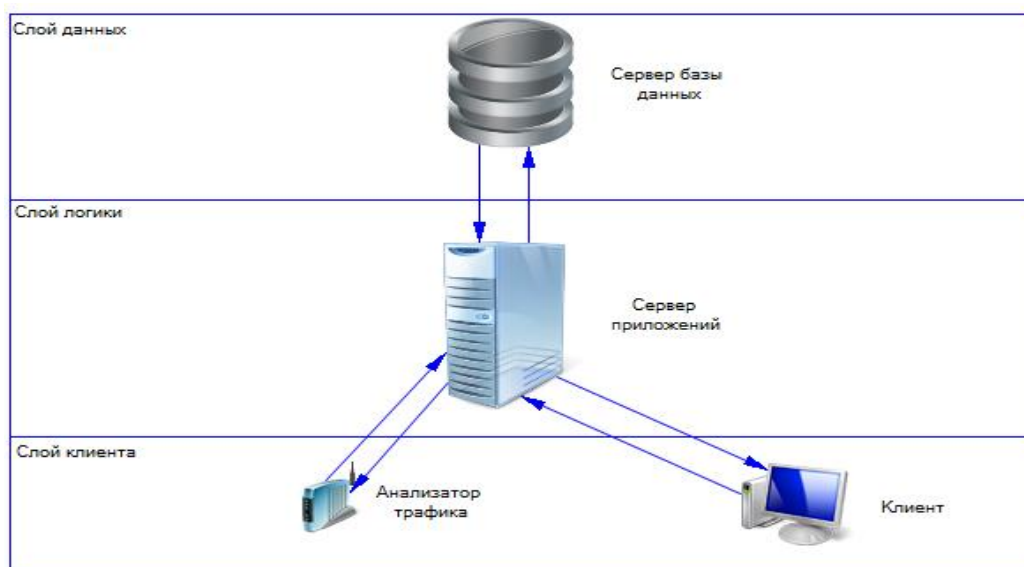


Рисунок 4 – Схема трёхуровневой архитектуры «клиент-сервер»

Слой клиента является уровнем интерфейса пользователя и приложения, главная задача которого заключается в предоставлении отправляемых на сервер команд и их результатов в понятном для пользователя виде (Рис. 5). Интерфейсы пользователя и анализатора трафика отличаются. Слой клиента содержит два интерфейса:

- 1) Интерфейс пользователя. Данный интерфейс позволяет отправлять на сервер необходимые запросы и отображать результаты запросов в понятном для пользователя виде. Запросы пользователя представлены в виде команд с параметрами, которые отправляются как набор специализированных операционных кодов, которые понятны только серверу приложений.
- 2) Интерфейс анализатора трафика. Данный интерфейс необходим лишь для обмена сообщениями и управляющими сигналами между сервером приложений и устройством. Используя этот интерфейс осуществляется контроль работы анализатора.

Слой логики выполняет координацию программы, обработку полученных операционных кодов, производит логические вычисления, осуществляет контроль подключенного анализатора USB-трафика. Контроль заключается в

том, что на устройство отправляются управляющие сигналы. При помощи управляющих сигналов, производится настройка анализатора и незамедлительное получение текущего буфера устройства (Рис. 5). Также используя определенные команды, можно осуществлять удаленный контроль путем отправки пакетов, где в роли получателя будет выступать исследуемая система. Эта особенность позволяет также осуществлять корректировку передаваемых данных в целях исправления ошибок, а также для проведения отладки исследуемой системы анализатором трафика.

Слой данных используется для хранения, передачи и обработки информации. Информация по запросу от сервера приложений извлекается из базы данных и отправляется в слой логики для обработки. В конечном счете обработанные с помощью слоя логики данные отправляются пользователю.

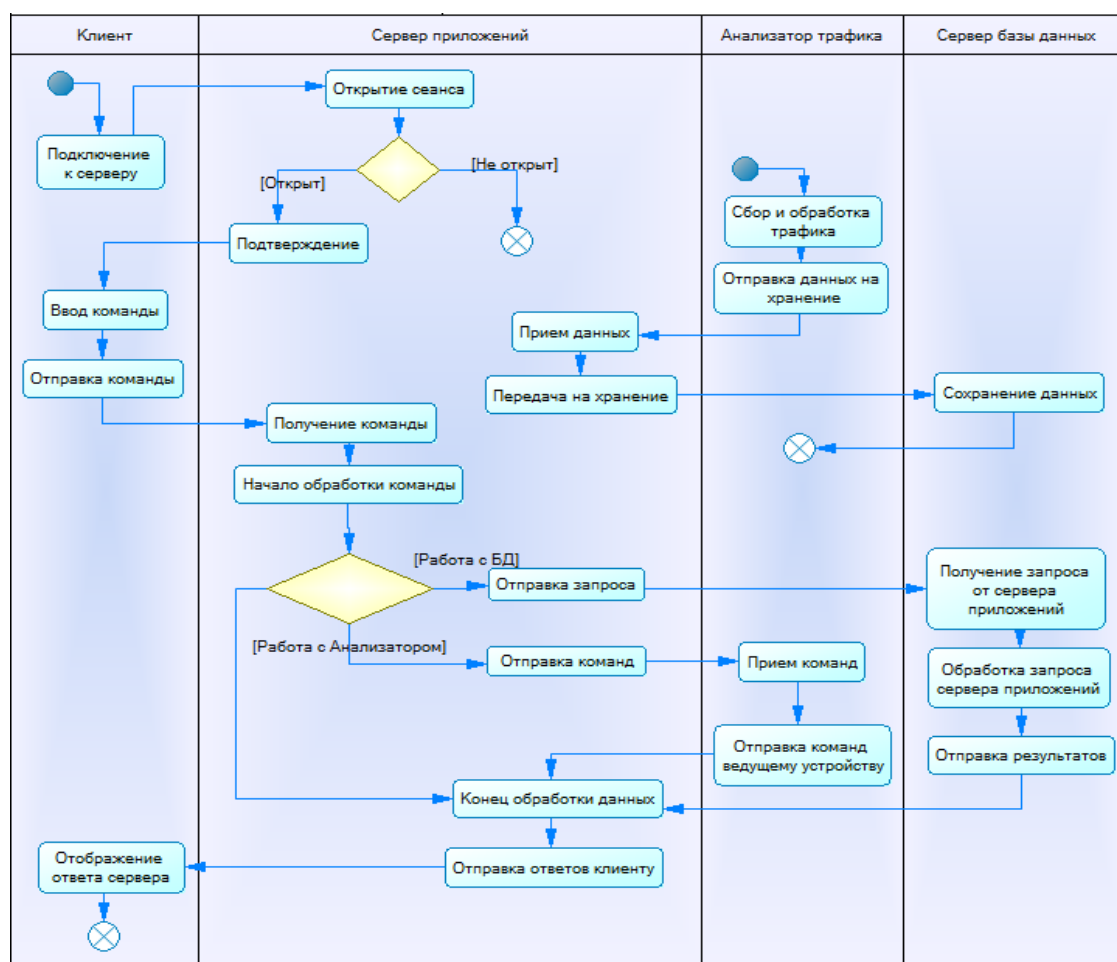


Рисунок 5 - Диаграмма деятельности трёхуровневой архитектуры «клиент-сервер»

2.2.2 Взаимодействие средствами сотовой связи

Для полноценной работы устройства анализа трафика необходимо подключение к серверу приложений. Следует иметь в виду то, что исследуемая система может представлять из собой совершенно любое устройство, которое имеет интерфейс USB. Исходя из этих особенностей, подключение исследуемого объекта к вычислительной сети необязательно и, соответственно, архитектура этой системы может иметь собственную специфику. В качестве таких систем могут выступать устройства, не имеющие операционной системы, к которым по интерфейсу USB можно подключить другие устройства. А если устройство не имеет операционной системы, то, к сожалению, возможность отследить трафик отсутствует. Выходом является отправка полученной информации на удаленный сервер на хранение, где в дальнейшем эта информация будет обработана.

Подводя итог, получается, что наиболее подходящий способ передачи информации, полученной с USB трафика - отправка информации с самого анализатора. Учитывая то, что при исследовании необходимой системы подключение к сети Интернет для работы с сервером может отсутствовать, а также учитывая то, что передача трафика на сервер осуществляется, не вмешиваясь в работу ведущего и ведомого устройств, то для взаимодействия с сервером приложений по сети Интернет подойдет пакетная радиосвязь общего пользования GPRS.

Для осуществления передачи данных используется GSM-модуль, и, соответственно, для работы модуля необходимо наличие SIM-карты. Используя отдельный модуль, анализатор трафика не зависит от исследуемой системы и не оказывает влияние на нее.

GSM-модуль позволяет взаимодействовать с сервером, осуществляя пакетную передачу данных, средствами мобильной связи. При использовании пакетной радиосвязи общего пользования (GPRS) информация разбивается на пакеты и отправляется через голосовые каналы, не используемые в данный момент. Данная технология подразумевает более эффективное использование

доступных ресурсов GSM-сети. Стоит отметить, что при этом приоритет передачи трафика (как голосового, так и данных) выбирается исключительно оператором связи [6].

Абоненту для передачи данных предоставляется отдельный канал, который используется для передачи голоса, с использованием модема, встроенного в мобильный терминал. Канал занят на все время, пока осуществляется передача информации через него. Пакетная радиосвязь общего пользования – это система, поддерживающая и реализующая протокол пакетной передачи данных в рамках сотовой связи сети GSM [7].

При использовании GPRS системы данные располагаются по пакетам и передаются в эфир. Стоит отметить то, что применение нескольких голосовых каналов одновременно позволяют достичь при передаче информации высокую скорость. При этом процесс установки соединения может занимать порядка нескольких секунд. Именно в этом режим пакетной передачи данных имеет принципиальное отличие. В результате у абонента имеется возможность осуществить передачу данных, не занимая каналы в интервалы времени между передачей информации. Данная особенность дает возможность использовать ресурсы сети эффективнее [8].

2.3 Разработка аппаратной части

Для реализации функционала, описанного в разделе «1.2.1. Сбор и обработка информации», а также опираясь на диаграмму действий, описывающую взаимодействие с анализатором трафика (Пункт «2.1. Анализ трафика», Рис. 3), потребуются следующие составные модули и контроллеры:

- аппаратная платформа;
- модуль для работы с USB-интерфейсом с защитой от перегрузок;
- GSM-модуль.

Главная особенность аппаратной части заключается в ее сравнительно небольшом размере, в наличии только самых необходимых возможностей (т.е. без избыточного функционала).

2.3.1 Анализ аналогов аппаратных платформ

Существует множество простых аппаратных платформ, предназначенных для решения одних и тех же задач. Для начала разработки аппаратной части работы, которая выполняет основные задачи анализатора трафика, а именно: сбор, обработку, отправку на хранение удаленному серверу проходящего по каналу связи трафика между ведущим и ведомым USB-устройствами, необходимо произвести сравнительный анализ самых типовых платформ.

Для сравнения рассматривались следующие аппаратные платформы:

- Семейство микроконтроллеров STM32F1x: STM32VLDISCOVERY.
- Платы Raspberry Pi (model B) – одноплатный компьютер.
- Плата семейства Arduino: Arduino Leonardo.

Данные платы выбраны потому что именно эти они являются максимально близкими по функциональным возможностям.

2.3.1.1 Платформа STM32VLDISCOVERY

Особенность данных оценочных плат заключается в завершенном решении для старта разработки программного обеспечения на микроконтроллерах – сам микроконтроллер с необходимой обвязкой и внешними компонентами, а также интегрированный программатор-отладчик ST-Link. В основе STM32VLDISCOVERY заложен 24 МГц микроконтроллер ARM Cortex-M3 с 8Кб оперативной памяти, 128 Кб внутренней памяти, многофункциональными таймерами, аналоговой периферией и разнообразными последовательными интерфейсами обмена данных (Рис.6). Напряжение питания 3.3В (2.7 - 3.6), все пины 5В-толерантны. На плате есть стабилизатор напряжения 3.3В, который питается от внешнего напряжения 5В, или USB [9].

Заметим, программатор/отладчик ST-Link можно использовать в качестве отдельного программатора для внешних устройств. Все выводы микроконтроллера доступны на разъемах платы. Питание осуществляется через шину USB или от внешнего источника. На платах установлены: светодиоды – статусные (т.е. индицирующие состояние питания, обмена данными и т.д.) и пользовательские, кнопки сброса для задач пользователя. Кроме того, на ряде плат имеются датчики - акселерометры, гироскопы, микрофоны и др., а также средства визуализации, такие как TFT и E-Ink дисплеи, сегментные ЖКИ [10].

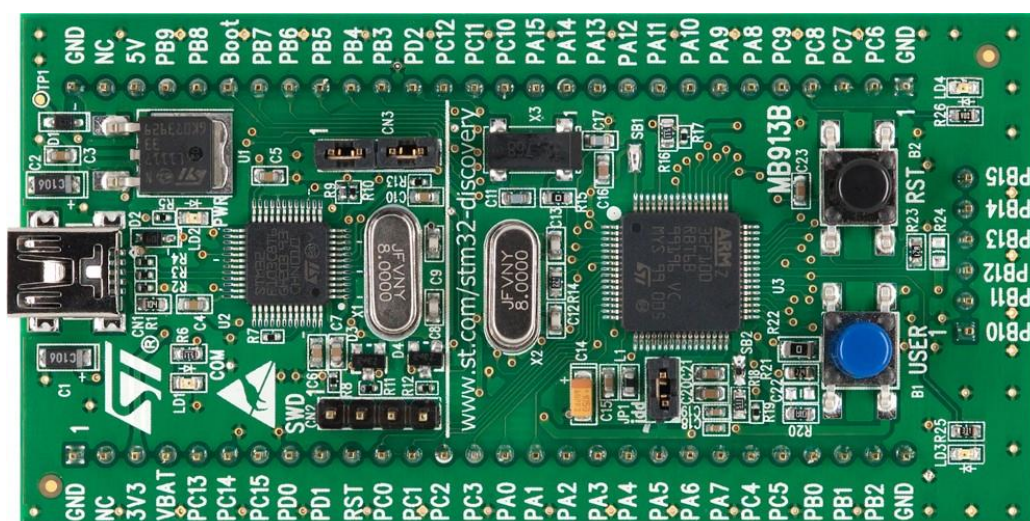


Рисунок 6 – Аппаратная платформа STM32VLDISCOVERY

2.3.1.2 Платформа Raspberry Pi (model B)

Raspberry Pi является полнофункциональным компьютером. Он обладает всеми атрибутами настоящего компьютера: выделенным процессором, памятью и графическим драйвером для вывода через HDMI. На нем даже работает специальная версия операционной системы Linux. Поэтому на Raspberry Pi легко установить большинство программ для Linux. Raspberry Pi можно использовать как полноценный медиа-сервер, или сервер приложений (например, почтовый сервер, сервер печати и т.д.).

Хотя в данной аппаратной платформе и отсутствует внутреннее хранилище данных, на этом компьютере можно использовать смарт-карты в

качестве флэш-памяти, обслуживающей всю систему. Таким образом, можно быстро выгружать для отладки различные версии операционной системы, или программных обновлений. Поскольку это устройство обеспечивает независимую связь по сети, его можно настраивать и для доступа по SSH, и пересылать на него файлы по протоколу FTP [11].

Данная аппаратная платформа содержит BCM2837 с 4 ядрами Cortex-A7 с частотой 1ГГц, оперативную память размером 1ГБ и полноценный Linux, основанный на Debian. Данные возможности помогут решить множество задач, требовательных к вычислительным ресурсам. Среди них можно выделить компьютерное зрение, обработку звука в реальном времени, создание веб-сервисов.

К Raspberry Pi можно подключить различную аппаратуру используя порты HDMI, USB. Модули BLE и WiFi на борту помогут соединить компьютер с другими устройствами без проводов. На плате нет АЦП, поэтому подключение аналоговых сенсоров возможно только с помощью внешних, дополнительных компонентов. Предоставляется лишь 1 аппаратный ШИМ-канал, что усложняет работу с периферией, которая управляется широтно-импульсной модуляцией [12]. Внешний вид платформы представлен на Рис. 7.

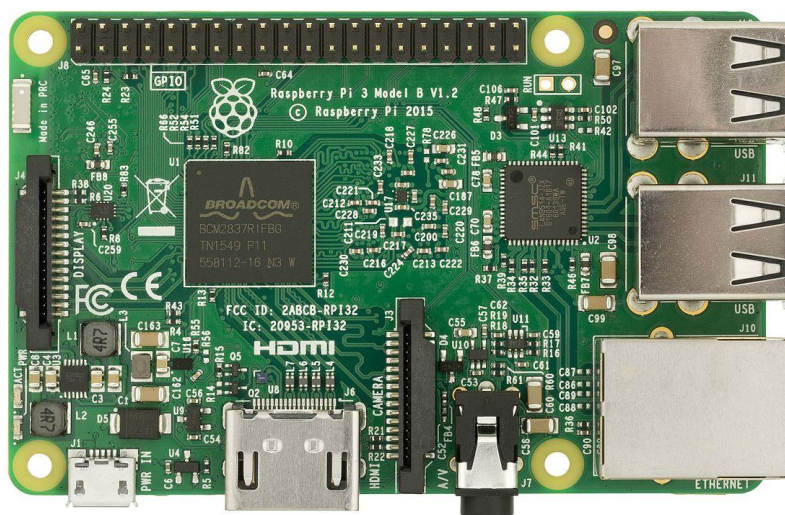


Рисунок 7 – Аппаратная платформа Raspberry Pi (model B)

2.3.1.3 Платформа Arduino Leonardo

Arduino Leonardo - это устройство на базе микроконтроллера ATmega32U4 (Рис. 8.). В его состав входит все необходимое для работы с данным микроконтроллером: 20 цифровых входов/выходов (7 из которых могут работать в качестве ШИМ-выходов, 12 - в качестве аналоговых входов), кварцевый резонатор на 16 МГц, разъем микро-USB, разъем питания, разъем для внутрисхемного программирования ICSP (In-Circuit Serial Programming) и кнопка сброса. Для начала работы с Leonardo достаточно просто подать питание от AC/DC-адаптера, или батареи, либо подключить его к компьютеру с помощью USB-кабеля.

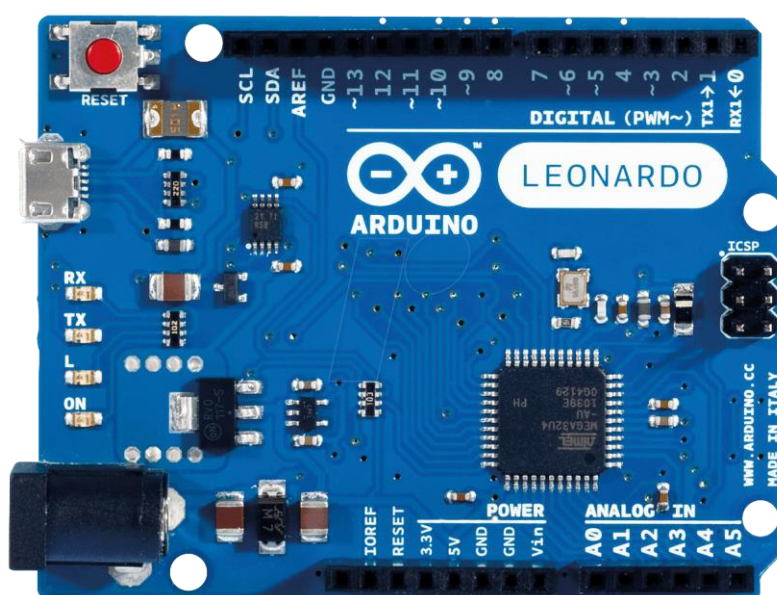


Рисунок 8 - Аппаратная платформа Arduino Leonardo

Отличие Leonardo от всех предыдущих плат заключается в том, что его USB-контроллер встроен непосредственно в микроконтроллер ATmega32U4, что исключает необходимость в дополнительном процессоре. Благодаря этому, при подсоединении к компьютеру, Leonardo может определяться не только как виртуальный (CDC) COM-порт, но и как обычное HID-устройство. Кроме того, такая архитектура оказывает влияние и на поведение платы.

Одно из преимуществ использования единого процессора для выполнения программ и USB-связи - это гибкость при взаимодействии устройства с другими

устройствами. В частности, благодаря этому, данная аппаратная платформа может определяться в системе не только, как виртуальный последовательный порт, используемый для прошивки и передачи данных, но и работать в качестве HID-устройства, эмулируя, например, клавиатуру, или мышь [13].

2.3.1.4 Сравнительная характеристика аппаратных платформ

В таблице 1 представлены краткие характеристики выбранных плат.

Таблица 1 – Технические характеристики аппаратных платформ

Платформа	STM32 VLDISCOVERY	Raspberry Pi (model B)	Arduino Leonardo
Ориентировочная цена	20\$	45\$	6\$
Максимальная длина	8,4 см	8,5 см	6,9 см
Максимальная ширина	4,3 см	5,4 см	5,4 см
Микроконтроллер	ARM	BCM2837	ATmega32u4
Тактовая частота	24 МГц	1200 МГц	16 МГц
ОЗУ	8 КБ	1 ГБ	2,5 КБ
Flash-память	128 КБ	SD карта	32 КБ
EEPROM	-	-	1 КБ
Рабочее напряжение	5 В	5 В	5 В
Минимальное энергопотребление	170 мА (0,85 Вт)	700 мА (3,5 Вт)	42 мА (0,3 Вт)
Цифровые линии ввода/вывода	18	8	20
Аналоговые входы	16	-	12
Каналы ШИМ	4	-	7
Вид доступа в Интернет	-	Ethernet (RJ-45), Wi-Fi	-
USB-разъём	mini USB	micro USB, USB 2.0 x4	micro USB

Анализируя технические характеристики, можно сделать вывод о том, что Arduino является самым бюджетной вариантом, к примеру, Arduino Leonardo на момент написания выпускной квалификационной работы стоит всего 6\$, в то

время как другие платы данного семейства стоят от 5\$ до 50\$ в зависимости от функционала и размеров конкретной платы. Стоит отметить, что стоимость датчиков, сенсоров и других внешних устройств не превышает 10\$. К тому же для работы с Raspberry Pi необходимо отдельно приобрести SD-карту, что увеличивает стоимость на 5-10\$.

Arduino Leonardo – малогабаритная плата, что делает ее более удобной для использования в небольших по размеру проектах. Стоит заметить, что при установке SD-карты Raspberry Pi превышает приведенные в таблице 3 размеры, что делает ее еще более непрактичной в использовании для конкретной задачи.

Для разработки приложений, требующих работы с USB-интерфейсом более практичны STM32VLDISCOVERY и Raspberry Pi. Посредством USB можно подключать различные модули, в том числе модули беспроводной передачи данных и реализовать функции беспроводной передачи данных и подключение к сети Интернет.

На платформе Arduino, как и на STM32VLDISCOVERY также можно реализовать приложения с поддержкой обмена данными по интерфейсу USB с помощью плат расширения (Shield), но данная функциональность таких приложений будет ограничена характеристиками, как аппаратной платформы, так и платы расширения. Для работы по сети можно подключить GSM-модуль для беспроводной передачи данных, вне зависимости от наличия сети в том месте, где будет использоваться устройство, главным условием будет являться наличие мобильной связи [14].

Так как главная особенность и преимущество Arduino Leonardo заключается в том, что его контроллер USB встроен непосредственно в основной микроконтроллер ATmega32U4, это дает возможность реализации задачи обмена данными по интерфейсу USB. В частности, благодаря этому, данная платформа позволяет гибко работать с ведущим устройством при их взаимодействии друг с другом. Используя виртуальный последовательный порт, для работы в качестве HID-устройства, появляется возможность реализовать работу

программируемого USB-хаба – осуществлять ретрансляцию входящих пакетов с одного USB-порта в другой, а также проводить копирование данных кадров, а затем и их обработку.

Arduino Leonardo – это функциональная и гибкая платформа разработки встраиваемых приложений с огромными возможностями для взаимодействия с различными устройствами. Данная аппаратная платформа позволит достичь поставленной цели и поставленных задач выпускной квалификационной работы (Пункт 1.1 Цели и задачи).

2.3.2 Анализ аналогов модулей сотовой связи

GSM/GPRS модуль представляет собой беспроводное коммуникационное устройство (модем) для приема/передачи данных в сетях мобильной связи. В любом мобильном телефоне установлен такой модуль, благодаря которому возможно голосовое общение, прием сообщений, выход в сеть Интернет [15].

GSM/GPRS модуль позволит взаимодействовать с сервером, осуществляя пакетную передачу данных, средствами мобильной связи. При использовании GPRS информация собирается в пакеты и передаётся через не используемые в данный момент голосовые каналы. Для осуществления передачи данных, используя GSM-модуль, необходимо наличие SIM-карты. Используя отдельный модуль, анализатор трафика не зависит от исследуемой системы и не оказывает влияние на нее.

Для начала разработки аппаратной части работы, одной из основных задач которой является отправление на хранение удаленному серверу, проходящего трафика по каналу связи USB, необходимо определить какой модуль лучше всего использовать. Стоит отметить, что критерием выбора являются следующие характеристики и возможности каждого модуля:

- стоимость модуля не более 10\$;
- низкое энергопотребление;
- длина и ширина модуля не более 3 см;

- возможность работы со стеком протоколов TCP/IP.

Для сравнения рассматривались следующие модули, удовлетворяющие критериям представленных выше:

- GSM/GPRS модуль SIM800L.
- GSM/GPRS модуль Ai-Thinker A6.

Данные платы выбраны потому что именно эти платы являются максимально близкими по возможностям решения однотипных задач.

2.3.2.1 GSM/GPRS модуль SIM800L

SIM800L бюджетный широко функциональный GSM/GPRS + Bluetooth модуль компании SIMCom Wireless Solutions (Рис. 9). В данный модуль имеет на борту Bluetooth трансивер, поддерживает GPRS multi-slot class 12, обеспечивая вдвое большую скорость выгрузки, а также обладает небольшим энергопотреблением. SIM800 имеет PCM-интерфейс и аналоговый аудио интерфейс. Базовая прошивка модуля поддерживает функции декодирования DTMF-тонов, записи аудиофайлов и работу с MMS. Bluetooth трансивер в данном модуле реализован на уровне чипсета.

SIM800L дает возможность работы с СМС, звонками, передачи данных GPRS. Управлять модулем можно как с компьютера (через USB-UART TTL конвертер), так и через аппаратные платформы семейств STM32, Arduino, Raspberry Pi. Взаимодействие с операторами сотовой связи зависит от прошивки модуля [16].

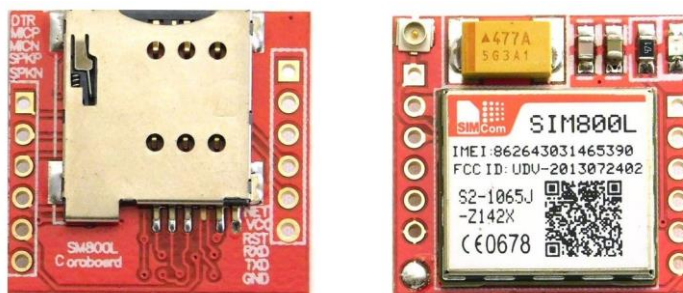


Рисунок 10 - GSM/GPRS модуль SIM800L

2.3.2.2 GSM/GPRS модуль Ai-Thinker A6

Ai-Thinker A6 бюджетный функциональный GSM/GPRS модуль компании ShenZhen Ai-Thinker CO (Рис. 10). Данный модуль имеет на борту UART интерфейс, что позволяет осуществлять взаимодействие с различными контроллерами, аппаратными платформами, модулями, компьютерами. Также имеет функции, позволяющие осуществлять звонки и принимать звонки от абонентов сотовых сетей, отправлять и принимать SMS-сообщения, передавать данные, используя пакетная радиосвязь общего пользования. Имеет PCM-интерфейс и аналоговый аудио интерфейс соответственно.

Особенность данного модуля заключается в том, что он может работать в четырех диапазонах сети (850 / 900 / 1800 / 1900 МГц) что позволяет использовать любого имеющегося оператора сотовой связи вне зависимости от прошивки модуля где угодно, в том числе на территории Российской Федерации и стран СНГ [17].

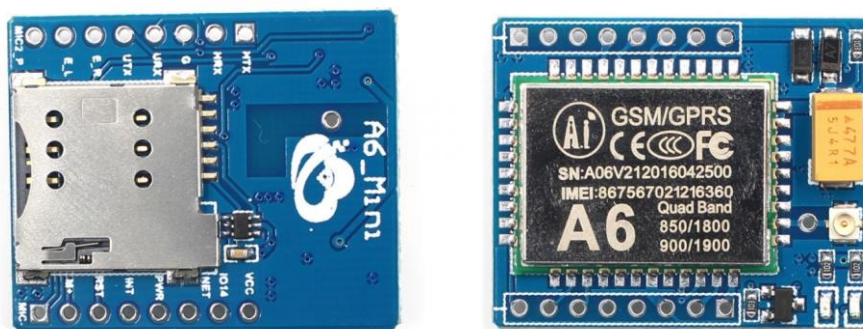


Рисунок 10 - GSM/GPRS модуль Ai-Thinker A6

2.3.2.3 Сравнительная характеристика GSM/GPRS модулей

В таблице 2 представлены краткие характеристики выбранных GSM/GPRS модулей.

Анализируя технические характеристики, выходит, что Ai-Thinker A6 является наиболее подходящим и, при этом, бюджетным вариантом. Данный модуль, может работать в четырех диапазонах сети, что позволяет использовать

любого имеющегося оператора сотовой связи в любой точке мира, в том числе на территории Российской Федерации и стран СНГ.

Таблица 2 – Технические характеристики GSM/GPRS модулей

Платформа	SIM800L	Ai-Thinker A6
Ориентировочная цена	3,5\$	3,5\$
Максимальная длина	2,5 см	3,3 см
Максимальная ширина	2,5 см	2,4 см
Класс передачи данных GPRS	GPRS class 12	GPRS class 10
Диапазоны GSM/GPRS	850 / 900 / 1800 / 1900 МГц	850 / 900 / 1800 / 1900 МГц
Связь с устройствами	UART, Bluetooth	UART
Встроенные стеки протоколов	PPP, TCP/IP	PPP, TCP/IP, UDP/IP
Формат SIM-карты	micro-SIM	micro-SIM
Рабочее напряжение	4,4 В	3,7 В
Минимальное энергопотребление	500 мА (2,2 Вт)	100 мА (0,37 Вт)
Максимальное энергопотребление	1000 мА (4,4 Вт)	900 мА (3,33 Вт)

По сравнению с Ai-Thinker A6, модуль SIM800L не обладает такой возможностью, в связи с ограничением установленным производителем. Для использования модуля в различных регионах необходимо устанавливать соответствующую для данного региона версию прошивки. Также следует отметить, что все модули имеют режим автоматической регистрации в сети оператора. Данный процесс самый энергоемкий, а так как устройство использует питание, предоставляемое аппаратной частью исследуемой системой, то необходимо учитывать максимальное значение силы тока, предоставляемое устройству, равно 900 мА. Поскольку пиковое значение силы тока SIM800L в 1000 мА наблюдается в момент регистрации в сети оператора, то использования данного источника питания будет недостаточно, и тогда придется использовать сторонний источник питания, что накладывает некоторые ограничения и

увеличивает затраты при использовании устройства с данным модулем, что является нежелательным.

Ai-Thinker A6 – это функциональный и гибкий модуль для разработки встраиваемых приложений, осуществляющих взаимодействия с различными сервисами по сети Интернет и/или по сети GSM. Использование данного модуля позволит достичь поставленной цели и задач выпускной квалификационной работы (Пункт 1.1 Цели и задачи).

2.3.3 Плата расширения USB Host Shield 2.0

Анализ трафика данной выпускной квалификационной работы заключается в наблюдении за потоком данных, проходящим по каналу связи USB между ведущим и ведомым устройствами. Процесс сбора информации производится путем захвата трафика, проходящего через данное соединение исследуемой системы.

Аппаратная платформа Arduino Leonardo имеет всего лишь один разъем интерфейса USB, что для данной работы недостаточно. Для решения данной проблемы существуют специализированные платы расширения.

USB Host Shield 2.0 (Рис. 11) для Arduino позволяет подключить USB устройство к данной аппаратной платформе. Для этой платы расширения существует достаточно большая библиотека – USB Host Library, которая предоставляет разработчику интерфейс прикладного программирования (API) предоставляющая функционал, для поддержки различных USB устройств, тем самым инструментарий библиотеки позволяет работать со многими классами устройств.

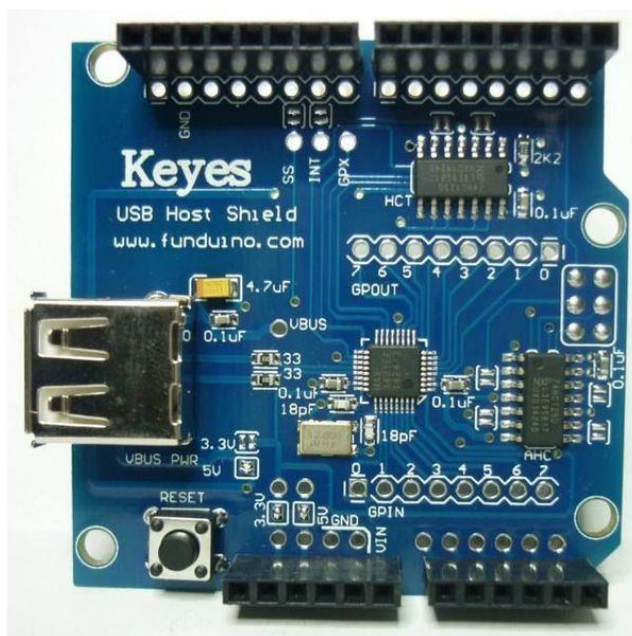


Рисунок 11 - Плата расширения USB Host Shield 2.0

Характерные особенности использования USB Host Shield 2.0 позволяют осуществить:

- поддержку и эмуляцию устройств, работающих на интерфейсе USB для взаимодействия с человеком (HID-устройства);
- поддержку и эмуляцию устройств, работающих на цифровом интерфейсе музыкальных инструментов (MIDI-устройства);
- взаимодействие с устройствами для хранения данных, например, USB флэшки, внешние жесткие диски;
- взаимодействие с устройством используя последовательные порты - FTDI, PL-2303, ACM;
- взаимодействие с устройствами Bluetooth, Bluetooth HID, Bluetooth SPP.

Данная плата расширения совместима со всеми аппаратными платформами семейства Arduino. USB Host Shield основан на микросхеме программно-управляемого USB-контроллера Max3421E.

Обмен данными между микросхемой Max3421E и микропроцессором ATmega осуществляется по SPI интерфейсу. Схема питания USB Host Shield 2.0 осуществляет подачу 3,3В на микросхему Max3421E и согласование сигнальных

уровней напряжений 5В микросхемы ATmega на плате Arduino. Также данная плата расширения обеспечивает питание периферийных USB устройств постоянным напряжением 5В. На рисунке 12 продемонстрирован вид Arduino Leonardo с подключённой платой расширения [18].

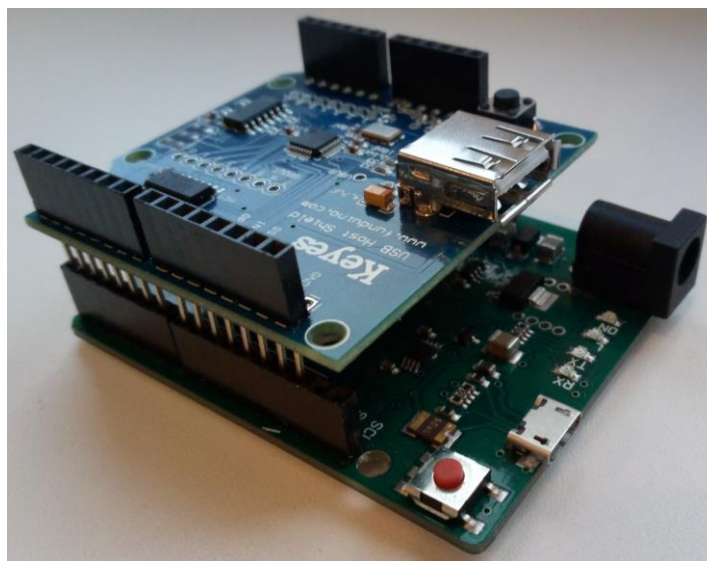


Рисунок 12 – Arduino Leonardo с подключённой платой расширения USB Host Shield 2.0

2.4 Схема подключения

Перед началом разработки программного обеспечения, необходимо создать макет устройства, осуществляющего анализ USB-трафика (Рис. 13). Макет является полноценным работоспособным устройством, но не имеет паяк и может быть разобран, или изменен в любой момент без каких-либо сложностей. Использование макета устройства дает возможность проверить корректность и стабильность работы разрабатываемого устройства, а также позволяет проверить работу программного обеспечения для микроконтроллера.

Для сборки макета необходимо опираться на сведения, представленные на графическом изображении, определяющие состав элементов и связи между ними (Рис. 14).

Данное графическое изображение получено при помощи бесплатного редактора Fritzing.

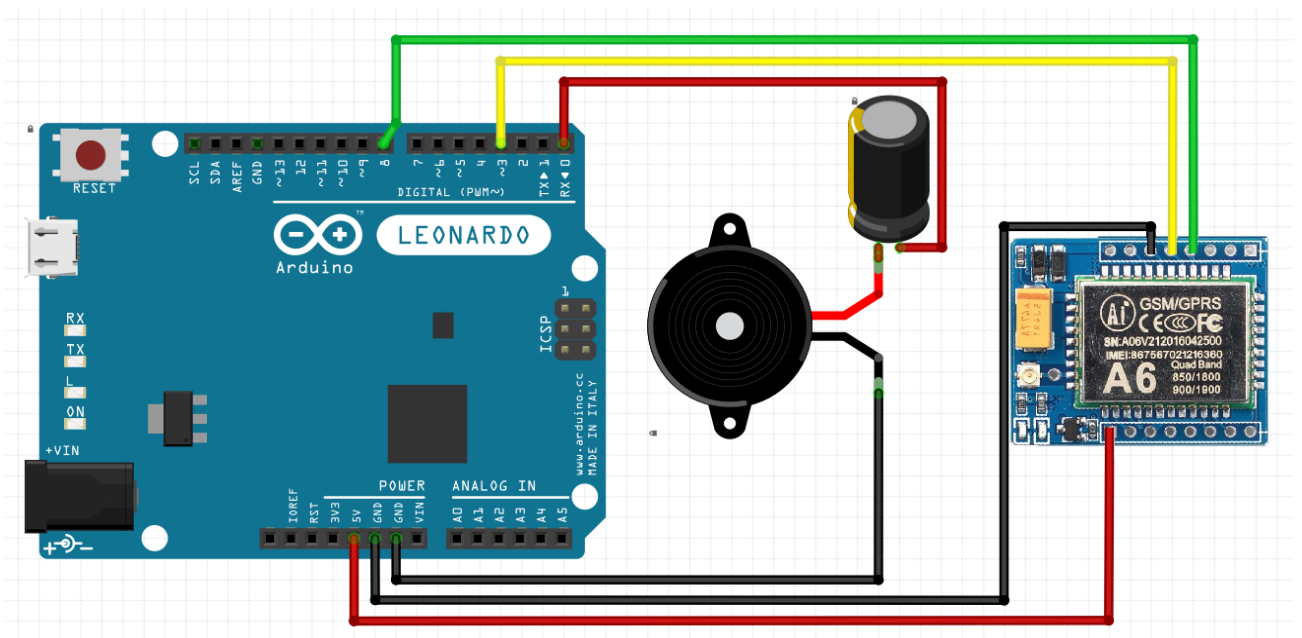


Рисунок 13 – Макет анализатора трафика USB

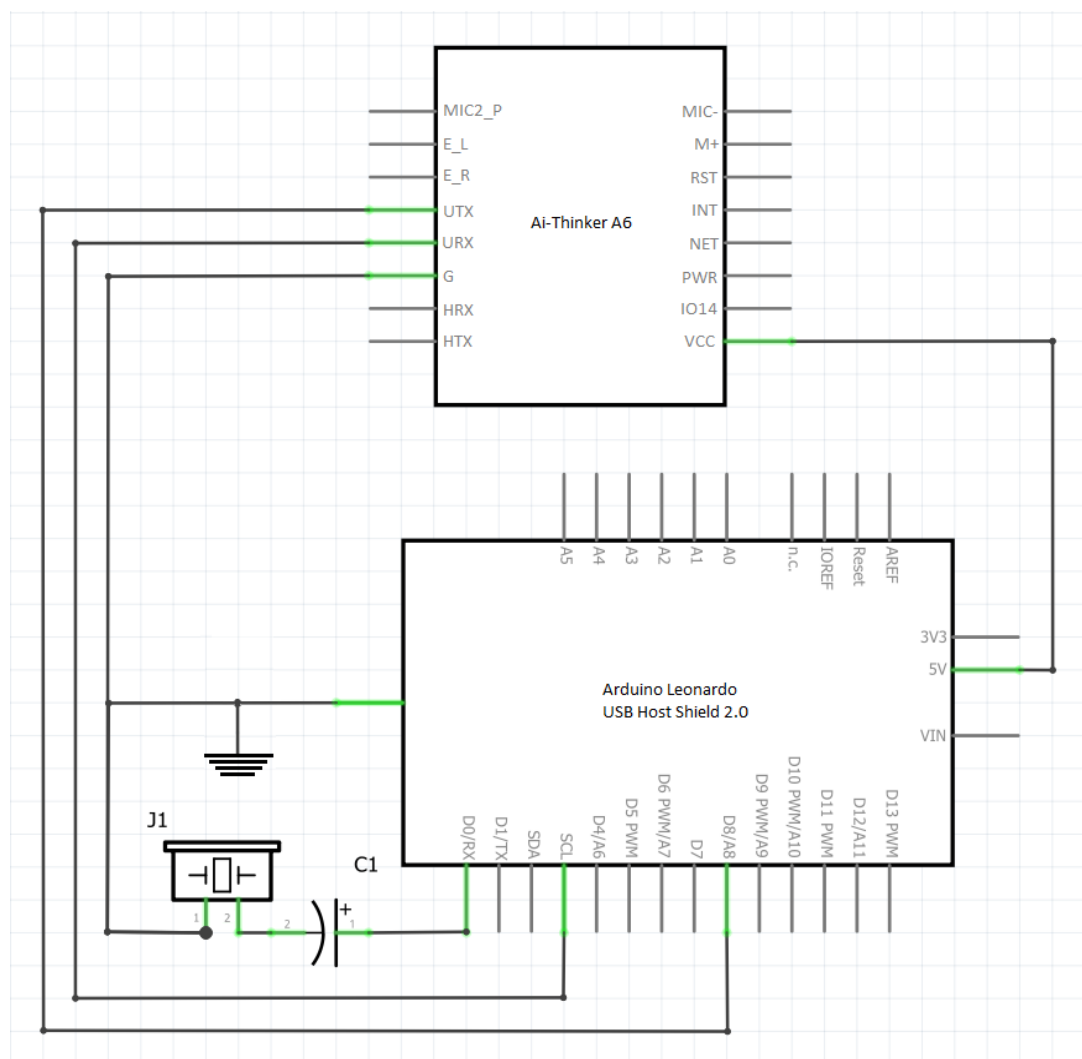


Рисунок 14 – Принципиальная схема устройства для анализа трафика USB

Список устройств, используемых в принципиальной схеме:

- Аппаратная платформа Arduino Leonardo.
- Плата расширения USB Host Shield 2.0.
- GSM/GPRS модуль Ai-Thinker A6.
- Спикер (J1).
- Конденсатор с ёмкостью 100 мкФ (C1).

USB Host Shield 2.0 подключается к Arduino Leonardo напрямую и не оказывает никакого влияния на предоставляемые линии аппаратной платформы, а лишь дополняет платформу, организуя работу интерфейсов с USB.

Внешний вид Arduino Leonardo с подключённой платой расширения USB Host Shield 2.0. представлен на рисунке 12 в пункте 2.4.1. Плата расширения USB Host Shield 2.0. Внешний вид собранного макета представлен на рисунке 15.

Преимущество использования макета заключается в том, что появляется возможность осуществлять разработку и отладку программного обеспечения для конкретной аппаратной части, вместо использования эмуляторов, которые не позволяют совершать тщательную отладку программной части.

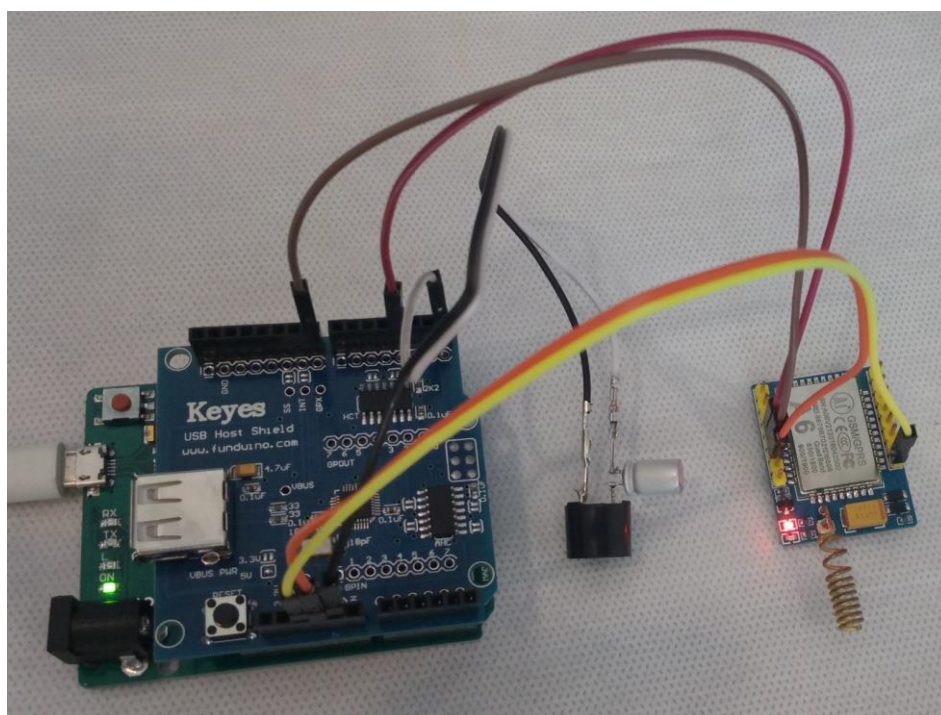


Рисунок 15 – Собранный макет анализатора трафика USB

2.5 Разработка программной части

Основными задачами данной выпускной квалификационной работы являются:

- 1) Проведение сбора, обработки, а также хранение информации на удаленном сервере.
- 2) Извлечение трафика, проходящего по каналу связи между ведущим и ведомым USB-устройствами.
- 3) Разработка клиент-серверного приложения.

Для того чтобы осуществить возможность без вмешательства в исследуемую систему отслеживать поток данных по каналу связи USB, и при необходимости изменять его, опираясь на спроектированную архитектуру «клиент-сервер» (Пункт «2.2.1. Архитектура «клиент-сервер»»), необходимо определить основные части системы и их функционал.

Система, осуществляющая работу с анализом трафика, содержит следующие части:

- Сервер приложений, осуществляющий:
 - координацию программы;
 - обработку полученных от клиента (как пользователя, так и анализатора трафика) данных;
 - логические решения и вычисления;
 - контроль за подключенным анализатором трафика USB.
- Сервер базы данных, позволяющий работать с системой управления базой данных для осуществления хранения, передачи и обработки информации.
- Пользовательский клиент, предоставляющий интерфейс пользователя и приложения, главная задача которого заключается во взаимодействии с инструментарием и возможностями сервера приложений и демонстрации результатов работы.

- Устройство, осуществляющее наблюдение за потоком данных, проходящим по каналу связи между ведущим и ведомым устройствами, а также осуществляющее взаимодействие с сервером приложений. Взаимодействие с сервером осуществляется посредством программного интерфейса, который необходим для обмена сообщениями, управляющими сигналами между сервером приложений и устройством. С использованием этого интерфейса производится контроль работы устройства.

2.5.1 Серверная часть

Объекты, входящие в состав информационной системы, не являются равноправными. Часть из них осуществляет контроль ресурсов (в том числе и хранение), другая часть просто имеет доступ к этим ресурсам. Соответственно следует отделять серверную часть, в которой заложена основная работа с логикой работы, от клиентской части, которая в свою очередь позволяет непосредственно работать с этой логикой.

Серверная часть состоит из двух основных компонентов:

- сервер базы данных;
- сервер приложений.

Основная идея разделения заключается в том, чтобы логика обработки данных и логика взаимодействия пользователей системы не пересекались с задачей хранения, предоставления информации.

2.5.1.1 Сервер базы данных

В данной выпускной квалификационной работе в качестве реляционной системы управления базой данных выступает MySQL компании Oracle, распространяемая под лицензией свободного программного обеспечения GNU General Public License. MySQL является решением для малых и средних приложений, так как данная система управления базой данных является частью системы «клиент-сервер», содержащей многопоточный SQL-сервер, который

поддерживает различные платформы, несколько клиентских программ и библиотек, инструменты администрирования и широкий диапазон программных интерфейсов приложений (API-интерфейсов), используя которые достигается простота и скорость в работе [19].

Сервер базы данных выполняет обслуживание и управление базой данных и отвечает за целостность и сохранность данных, а также обеспечивает операции ввода-вывода при доступе к информации. Данный сервер относится к слою данных [20].

Основная задача слоя данных заключается в извлечении необходимой информации из базы данных по запросу от сервера. Извлеченные данные отправляются сервером базы данных непосредственно заказчику информации. В случае записи, удаления или изменения информации, хранящейся в базе данных, сервер отправляет заказчику результат произведенной операции.

Поскольку разрабатываемое устройство отслеживает трафик между ведущим и ведомым устройствами, а также отслеживаемый трафик обрабатывается и отправляется на хранение в базу данных, для дальнейшей обработки данных, то в базе будет храниться следующий набор сведений о трафике:

- Сообщение. Данная информация содержит данные, полученные в результате работы анализатора трафика.
- Дата создания записи в базе данных. Данная информация необходима для определения времени регистрации проанализированного трафика в базе.
- Состояние «Запись не прочитана». Так как поток данных может быть большим, то необходимо определять прочитанные и непрочитанные пользователем записи. С использованием информации о данном состоянии появляется возможность осуществлять фильтрацию прочитанных данных.

Состояние «Запись удалена». Данная информация позволяет восстановить необходимый набор записей (отменить удаление), предотвратив безвозвратную потерю данных по вине пользователя.

С опорой на сведения, представленные выше была разработана концептуальная модель структуры базы данных (См. Рис. 16). Модель структуры базы данных содержит две сущности:

- Логгер – содержит информацию о полученных данных в результате анализа трафика.
- Состояние – содержит информацию о флагах состояния Логгера.

Между данными сущностями установлена связь один к одному, которая показывает, что Логгер обязательно должен иметь Состояние, а Состояние может иметь 0 или 1 Логгер.

Используя полученную диаграмму «сущность-связь» можно получить физическую модель базы данных (Рис. 17). Данная модель базы данных определяет способы размещения информации в среде хранения, а также способы доступа к этой информации. Физическая модель позволяет получить код запроса на языке SQL для нужной базы данных при помощи программных средств, решающих задачи моделирования данных (например, StarUML, Sybase PowerDesigner).

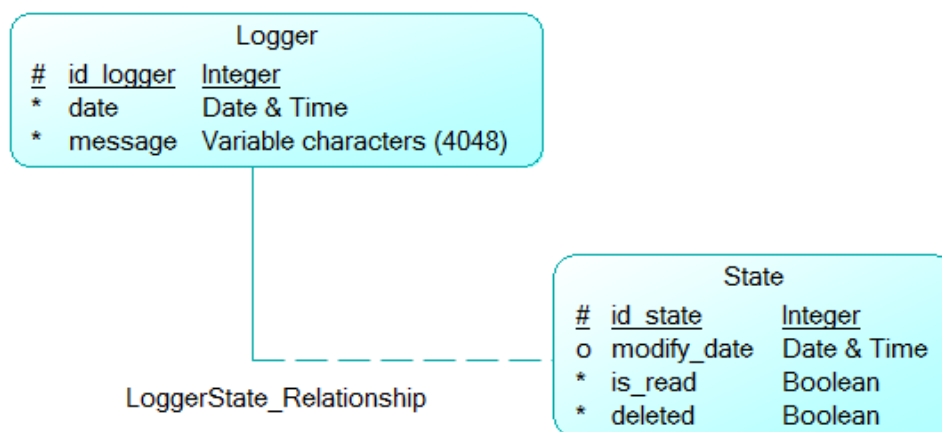


Рисунок 16 – Диаграмма «сущность-связь»

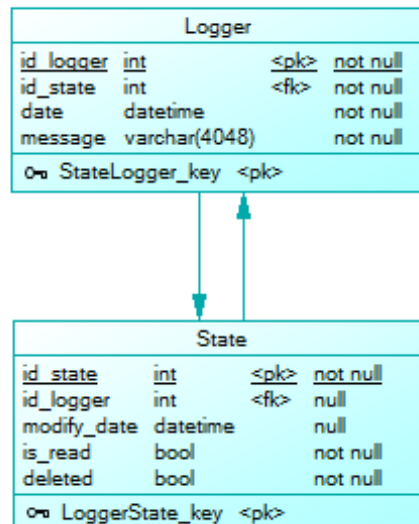


Рисунок 17 – Физическая модель данных

2.5.2.1 Сервер приложений

Запросы пользователя представлены в виде команд с параметрами, которые отправляются в виде специализированных операционных кодов, понятных только серверу приложений. Сервер осуществляет взаимодействие и работу с основной логикой:

- обрабатывает полученные операционные коды;
- осуществляет логические вычисления;
- осуществляет контроль подключенного анализатора USB-трафика;
- возможность работы с множеством клиентов одновременно.

Суть контроля заключается в том, что на устройство, которое выполняет анализ трафика, отправляются управляющие сигналы. При помощи управляющих сигналов производится настройка анализатора и незамедлительное получение текущего буфера устройства. Также используя определенные команды, можно осуществлять удаленный контроль путем отправки пакетов, где в роли получателя будет выступать исследуемая система.

Для организации логики работы сервера приложений применяется модульное программирование. Суть модульного программирования заключается в том, что программная реализация состоит из некоторой

совокупности самостоятельных блоков, именуемых как «модули». Использование данного вида программирования дает возможность упростить процесс тестирования программного обеспечения и, соответственно, обнаружение ошибок. Одна из особенностей данного метода заключается в том, что задачи, которые зависят от аппаратного обеспечения, могут быть реализованы отдельно от других задач, что позволяет улучшить мобильность программного обеспечения.

Программный модуль представляет из себя функционально законченный фрагмент программного обеспечения (информационной системы). В объектно-ориентированном программировании модуль представляет из себя класс - некоторая структура со своими полями и функциями, которая оформляется в отдельных файлах. Всего два файла: заголовочный файл и файл исходного текста класса. Модули можно объединять между собой, а результат объединения модулей называется пакетом.

На рисунке 18 изображена структурная диаграмма, которая демонстрирует разделение структуры программного обеспечения на некоторые компоненты, между которыми определены связи. Данные связи называются зависимостями. Зависимость показывает, что один элемент предоставляет сервис, который необходим другому элементу.

В роли физических компонентов могут быть модули, пакеты, библиотеки, исполняемые файлы, и т.д. Компоненты связываются между собой через зависимости, когда производится соединение необходимого интерфейса одного компонента с интерфейсом другого.

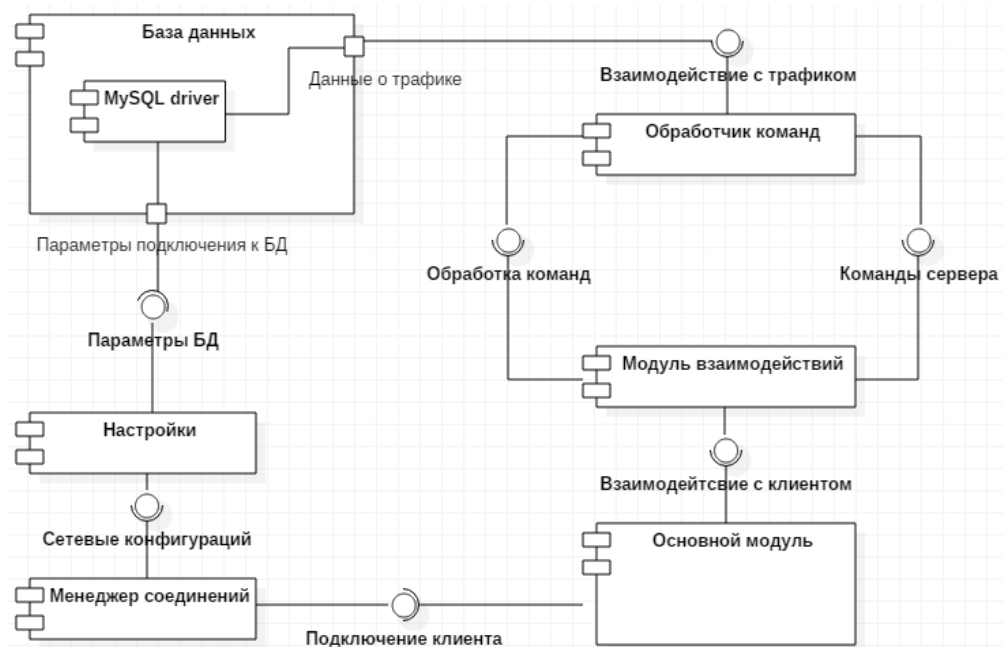


Рисунок 18 – Диаграмма компонентов сервера приложений

После моделирования структуры компонентов программы можно построить её диаграмму классов, содержащую структуру классов, их атрибуты, методы и связи между ними. (Рис. 19).

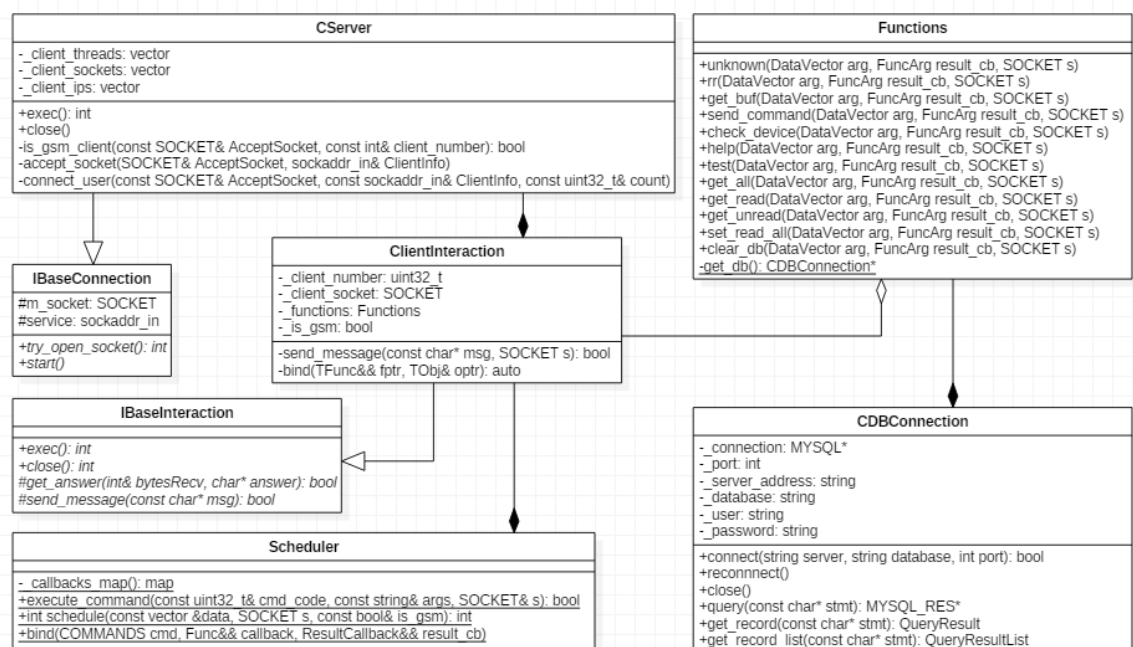
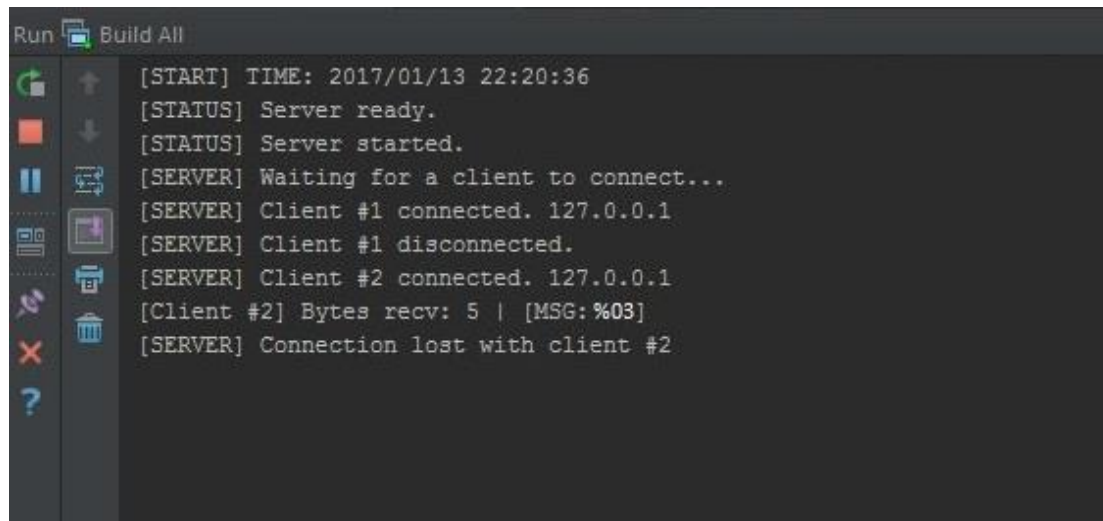


Рисунок 19 – Диаграмма классов сервера приложений

Сервер приложений является консольным приложением. В вывод консоли сервер будет помещать служебную информацию о входящих подключениях, о

полученных командах от различных пользователей, а также о статусе сервера. Это предоставляет возможность аудита всех событий происходящих на сервере приложений, благодаря которому можно отследить и выявить изменения в системе.

Пример аудита приведен на Рис. 20.

A screenshot of a terminal window with a dark background. On the left side, there is a vertical toolbar with various icons: a green play button, a red stop button, a blue double arrow, a blue square, a blue circle, a blue trash can, a blue 'X', and a blue question mark. The main area of the terminal displays a log of server events in white text. The log starts with a timestamp and then shows the server's status and interactions with two clients. The text is as follows:

```
Run Build All
[START] TIME: 2017/01/13 22:20:36
[STATUS] Server ready.
[STATUS] Server started.
[SERVER] Waiting for a client to connect...
[SERVER] Client #1 connected. 127.0.0.1
[SERVER] Client #1 disconnected.
[SERVER] Client #2 connected. 127.0.0.1
[Client #2] Bytes recv: 5 | [MSG:%03]
[SERVER] Connection lost with client #2
```

Рисунок 20 – Снимок интерфейса командной строки сервера приложения

2.5.2.1.1 Описание модулей сервера приложений

Для работы сервера приложений используются следующие модули:

- connection.h;
- server.h;
- interaction.h;
- clientinteraction.h;
- database.h;
- scheduler.h;
- functions.h;
- config.h.

Исходный код программного обеспечения сервера приложений представлен в Приложении В.

2.5.2.1.1.1 Менеджер соединений

Модуль `connection.h` – модуль базового интерфейса менеджера соединений. Данный модуль описывает основные поля и методы для реализации работы с входящими подключениями к серверу. Модуль содержит в себе абстрактный класс – `IBaseConnection`.

2.5.2.1.1.2 Основной модуль сервера

Модуль `server.h` – основной модуль сервера, который позволяет создать его объект. Основные поля и методы для реализации взаимодействия с входящими подключениями к серверу описаны в родительском классе `IBaseConnection`. Модуль содержит в себе основной класс для работы с сервером – `CServer`. Данный класс осуществляет работу сервера с использованием следующих методов:

- Функция `int try_open_socket()` - осуществляет инициализацию сокета и дальнейшее подключение клиента к серверу. Возвращает 0, если открытие сокета прошло успешно, иначе возвращает значение -1. Данный метод содержит в себе логику настройки сетевого подключения сервера.
- Функция `int exes()` - основной метод класса. Возвращает 0, если работа сервера завершена без аварий, иначе возвращает значение -1. Данный метод содержит в себе логику взаимодействия сервера с клиентами: производит обращение к менеджеру соединений для подключения клиента к серверу, а также к модулю взаимодействий для работы с клиентом (получение и передача информации, выполнение запросов клиента).
- Функция `void start()` - выполняет запуск серверной части: осуществляет инициализацию модулей, конфигурацию сетевого взаимодействия. Данный метод содержит логику работы с входящими соединениями позволяет начать работу с этими соединениями, и осуществлять обработку входящих запросов на подключение благодаря запуску методов

CServer::try_open_socket(), CServer::exec(), которые позволяют принимать и обрабатывать входящие подключения.

- Функция void close() - выполняет закрытие всех рабочих соединений с клиентами и завершение работы сервера.

2.5.2.1.1.3 Модуль взаимодействий

Модуль interaction.h – модуль базового интерфейса менеджера взаимодействий. Данный модуль описывает основные поля и методы для реализации взаимодействия с подключенными к серверу клиентами. Модуль содержит в себе абстрактный класс – IBaseInteraction.

Модуль clientinteraction.h – модуль менеджера взаимодействий. Данный модуль осуществляет взаимодействие с сетевыми клиентами (как с пользовательским клиентом, так и с клиентом анализатора трафика). Основные поля и методы для реализации взаимодействия подключенных к серверу клиентов описаны в родительском классе IBaseInteraction. Данный модуль содержит в себе основной класс для работы с клиентами – ClientInteraction.

Стоит отметить, что запросы пользователя представлены в виде команд с параметрами, которые на стороне кодируются и отправляются в виде специализированных операционных кодов, которые понятны только серверу приложений. Модуль обрабатывает полученные запросы клиента, приводя в соответствие операционному коду команды допустимую функцию. Для обработки команды менеджер взаимодействий обращается к модулю обработки команд.

Исходя из того, что анализатор трафика является клиентом, можно сделать вывод об отсутствии необходимости реализации отдельной логики для работы с анализатором трафика и для работы с обычным клиентом. Использование одного специализированного набора операционных кодов позволит, не меняя логики, анализатору трафика осуществлять взаимодействие с сервером приложений в качестве пользовательского клиента. Использование операционных кодов для

анализатора позволит осуществлять передачу серверу полученного трафика USB, а также принимать управляющие команды от клиента.

Для обработки запросов от клиентов сервера используются следующие методы:

- Функция `int exec()` - основной метод класса. Возвращает 0, если работа завершилась без аварий, иначе возвращает значение -1. Данный метод содержит логику взаимодействия между сервером и клиентом, осуществляет прием и передачу данных, обработку запросов пользователя. Для обработки команд клиента происходит обращение к модулю обработки команд. Результат обработки отправляется необходимому клиенту.
- Функция `void close()` – завершает работу модуля.
- Функция `bool send_message(const char* msg)` – метод, осуществляющий передачу информации клиенту. Параметр `msg` - массив отправляемых данных. Возвращает истину, если данные переданы успешно, возвращает ложь в случае возникновения неполадок при передаче информации.
- Функция `bool get_answer(int& bytesRecv, char* answer)` – метод, осуществляющий прием и первичную обработку ответа клиента. Параметр `bytesRecv` - количество полученных байт, `answer` - массив данных, содержащий ответ клиента. Возвращает истину, если данные получены успешно, возвращает ложь в случае возникновения неполадок при получении ответа.

2.5.2.1.1.4 Модуль базы данных

Модуль `database.h` – модуль взаимодействия с базой данных. Позволяет создать объект, который по заданным параметрам позволит работать с системой управления базой данных, используя драйвер клиента MySQL. Модуль содержит основную логику работы с сервером базы данных, позволяя отправлять запросы и получать результаты выполнения запросов в виде контейнера с необходимыми

данными. Модуль содержит в себе основной класс для работы с СУБД – CDBConnection. Данный класс упрощает работу с базой данных благодаря использованию следующих методов:

- Функция `bool connect(const std::string& server, const std::string& database, const unsigned int port)` – метод, осуществляющий подключение к базе данных MySQL. Данный метод содержит логику подключения к серверу базы данных, задачей которой является упрощение работы с хранилищем информации. Параметры `server` и `port` – содержат `ip`-адрес и порт, необходимые для подключения к серверу БД, `database` - имя конкретной базы данных. Возвращает истину, если удалось подключиться к указанному серверу, возвращает ложь в случае возникновения проблем с подключением.
- Функция `void reconnect()` - метод осуществляющий повторное подключение к базе данных MySQL, во избежание автоматического разрыва соединения с СУБД по истечению времени неактивности (более 5 минут) текущего соединения.
- Функция `void close()` – метод, осуществляющий завершение сеанса работы с базой данных MySQL.
- Функция `MYSQL_RES* query(const char* stmt)` – основной метод передачи текста запроса системе управления базой данных. Параметр `stmt` – текст SQL запроса. Данный метод содержит в себе логику передачи текста запроса и логику получения результата обработки запроса. Результат запроса - объект структуры `MYSQL_RES`, содержащей в себе имена столбцов и массив данных.
- Функция `QueryResult get_record(const char* stmt)` – метод, позволяющий получить по заданному запросу первую попавшуюся информацию из базы данных MySQL. Параметр `stmt` – текст SQL запроса. Результат запроса - вектор `QueryResult`, содержащий в себе имена столбцов и их значения.

- Функция `QueryResultList get_record_list(const char* stmt)` – метод, позволяющий получить по заданному запросу список необходимой информации из базы данных MySQL. Параметр `stmt` – текст SQL запроса. Результат запроса - вектор `QueryResultList`, каждый элемент которого является вектором значений `QueryResult`.

2.5.2.1.1.5 Модуль обработчика команд

Модуль `scheduler.h` – модуль обработчика команд. Модуль содержит в себе основной класс для работы с командами клиента – `Scheduler`. Данный класс позволяет создать объект осуществляющий обработку запросов клиента. Модуль обрабатывает запросы, полученные от клиента, приводя в соответствие полученному операционному коду команды функцию. В случае отсутствия операционного кода в справочнике допустимых кодов клиенту отправляется сообщение о том, что команда не найдена. Для организации работы обработчика команд используются методы:

- Функция `bool execute_command(const std::uint32_t& cmd_code, const std::string& args, SOCKET& s)` - метод обработки команд. Данный метод осуществляет поиск обработчика команды по коду и производит ее выполнение. Параметр `cmd_code` – операционный код, полученный от клиента, `args` – массив аргументов, необходимый для выполнения команды клиента, `s` – сокет клиента, для обратной связи. Метод возвращает истину, если для искомого операционного кода была найдена функция обработки, и исполнение команды прошло без ошибок, иначе возвращает ложь.
- Функция `int schedule(const std::vector<uint8_t> &data, SOCKET s, const bool& is_gsm)` – основной метод класса. Данный метод осуществляет анализ полученных от клиента данных – осуществляет разбиение полученных данных на операционные коды команд и их аргументы. Осуществляет первичный анализ полученных операционных кодов на их корректность, на доступ к ним того, или иного клиента, поскольку у клиента анализатора трафика имеется доступ не ко всем имеющимся

командам сервера. Параметр `data` – массив полученных от клиента байт, `s` – сокет клиента для обратной связи, `is_gsm` – флаг GSM-устройства (анализатора трафика). Метод возвращает 1, если команда выполнена успешно, возвращает 0, если не успешно, -1 – в случае возникновения ошибок в работе метода.

- Функция `void bind(COMMANDS cmd, Func&& callback, ResultCallback&& result_cb)` – метод связывающий операционный код команды, функцию обработки команд и результирующую функцию. Под результирующей функцией понимается функция, осуществляющая свою работу после исполнения функции обработки, т.е. функция, осуществляющая пост-обработку (к примеру отправку данных клиенту).

Для выполнения команды может потребоваться наличие параметров для функции обработчика. Обязательные параметры указываются непосредственно в соответствующей функции обработчика команд.

Реализация обработчиков команд расположена в классе `Functions` (`functions.h`). Каждый метод класса `Functions` принимает три параметра `arg` – вектор аргументов, необходимых для исполнения команды, `result_cb` – функция осуществляющая пост-обработку (отправку данных, например), `s` – сокет клиента, для обратной связи. Логика обработки команды заключается в том, что исполняется один из методов данного класса, по завершению которого вызывается функция пост-обработки, задачей которой является обработка результатов выполнения команды и отправка готовых данных клиенту.

Список команд с операционными кодами представлен ниже:

- Команда «`help`» – возвращает список доступных команд с их описанием. Для выполнения данной команды наличие параметров необязательно. Операционный код: «%01». Прототип функции обработки: `void Functions::help(DataVector arg, FuncArg result_cb, SOCKET s)`.

- Команда «device» – возвращает информацию о подключенном устройстве анализа трафика. Для выполнения данной команды наличие параметров необязательно. Операционный код: «%03». Прототип функции обработки: `void Functions::check_device(DataVector arg, FuncArg result_cb, SOCKET s)`.
- Команда «command {arg}» – отправляет управляющие воздействия на анализатор трафика. В ней обязательно наличие аргумента arg - вектор байт. Операционный код: «%04». Прототип функции обработки: `void Functions::send_command(DataVector arg, FuncArg result_cb, SOCKET s)`.
- Команда «clear_db» – осуществляет очистку базы данных. Очистка заключается в том, что всем выбранным записям таблицы Логгер присваивается признак удаления. Для выполнения данной команды наличие параметров необязательно. Операционный код: «%05». Прототип функции обработки: `void Functions::clear_db(DataVector arg, FuncArg result_cb, SOCKET s)`.
- Команда «get_all» - позволяет получить все существующие записи в базе данных, несмотря на значение флага состояния. После выполнения команды флаги состояния записей не изменяются. Для выполнения данной команды наличие параметров необязательно. Операционный код: «%06». Прототип функции обработки: `void Functions::get_all(DataVector arg, FuncArg result_cb, SOCKET s)`.
- Команда «get_buf» - позволяет в срочном порядке получить текущий буфер USB-трафика подключенного к серверу устройства. Для выполнения данной команды наличие параметров необязательно. Операционный код: «%07». Прототип функции обработки: `void Functions::get_buf(DataVector arg, FuncArg result_cb, SOCKET s)`.
- Команда «get_read» - позволяет получить все существующие записи в базе данных, которые не имеют флаг состояния «Запись не прочитана» и не имеют флаг «Запись удалена». После выполнения команды флаги состояния записей не изменяются. Для выполнения данной команды

наличие параметров необязательно. Операционный код: «%08». Прототип функции обработки: `void Functions::get_read(DataVector arg, FuncArg result_cb, SOCKET s)`.

- Команда «get_unread» - позволяет получить все существующие записи в базе данных, которые имеют флаг состояния «Запись не прочитана» и не имеют флаг «Запись удалена». После выполнения команды для записей выставляется флаг состояния «Запись прочитана». Для выполнения данной команды наличие параметров необязательно. Операционный код: «%09». Прототип функции обработки: `void Functions::get_unread(DataVector arg, FuncArg result_cb, SOCKET s)`.
- Команда «rr» – отправляет управляющее воздействие направленное на перезапуск анализатор трафика. Для выполнения данной команды наличие параметров необязательно. Операционный код: «%10». Прототип функции обработки: `void Functions::rr(DataVector arg, FuncArg result_cb, SOCKET s)`.
- Команда «set_read_all» - помечает все записи в базе данных флагом состояния «Запись прочитана». Для выполнения данной команды наличие параметров необязательно. Операционный код: «%11». Прототип функции обработки: `void Functions::set_read_all(DataVector arg, FuncArg result_cb, SOCKET s)`.
- Команда «store {arg}» - отправляет набор данных arg на хранение в базу данных. Данная команда доступна только для анализатора трафика. Для её выполнения наличие параметров обязательно. Операционный код: «%12». Прототип функции обработки: `void Functions::store(DataVector arg, FuncArg result_cb, SOCKET s)`.

2.5.2.1.1.6 Модуль настроек

Модуль `config.h` – модуль конфигураций сервера. Данный модуль содержит основные константы, основные параметры и сетевые настройки сервера приложений, а также содержит основные параметры для работы с

сервером базы данных. Модуль содержит в себе пространство имен – ServerCfg, в котором расположены все необходимые атрибуты.

2.5.3 Клиентская часть

Слой клиента является уровнем интерфейса пользователя и приложения, главная задача которого заключается в предоставлении отправляемых на сервер команд и результатов их выполнения в понятном для пользователя виде (Рис. 5). Стоит определить два клиентских интерфейса в слое клиента трехуровневой архитектуры «клиент-сервер»:

- Клиентский интерфейс пользователя.
- Клиентский интерфейс анализатора трафика.

Суть данного разделения заключается в том, что для анализатора необходимо определить другой интерфейс ввиду его функциональных особенностей. Интерфейс анализатора предоставляет возможность обмена сообщениями и управляющими сигналами между сервером приложений, и устройством, а интерфейс пользователя предоставляет, возможность пользователю самостоятельно взаимодействовать с сервером приложений. .

Клиент пользователя представляет собой программное обеспечение, которое предоставляет возможность пользователю осуществлять взаимодействие с сервером путем отправки запросов и вывода результатов запросов в понятном для пользователя виде. Запросы пользователя представлены в виде команд с параметрами, которые отправляются в виде специализированных операционных кодов, которые понятны только серверу приложений.

Для организации логики работы клиента пользователя применяется модульное программирование. На рисунке 21 изображена структурная диаграмма, которая демонстрирует разбиение программного обеспечения на некоторые структурные компоненты и связи между ними.

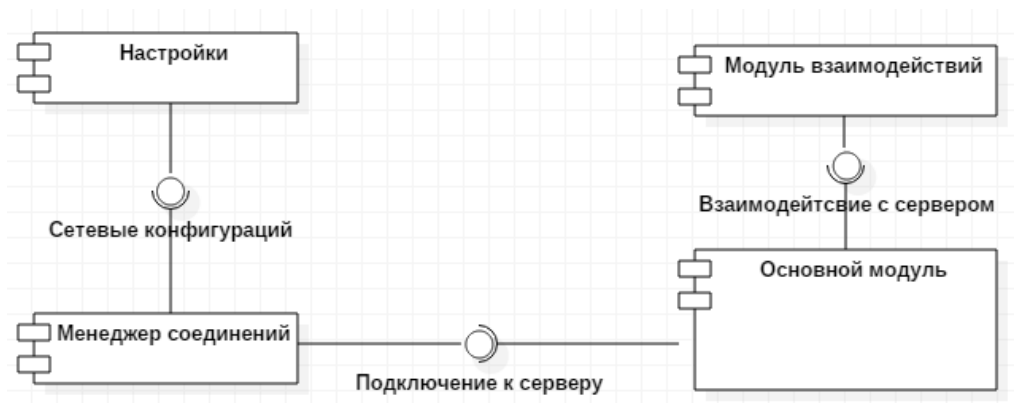


Рисунок 21 – Диаграмма компонентов клиента пользователя

Как и сервер приложений, клиент имеет модуль менеджера соединений и модуль взаимодействий. Определив компоненты, можно построить диаграмму классов, отражающую информацию о классах, их атрибутах, методах и связях между ними (Рис. 22).

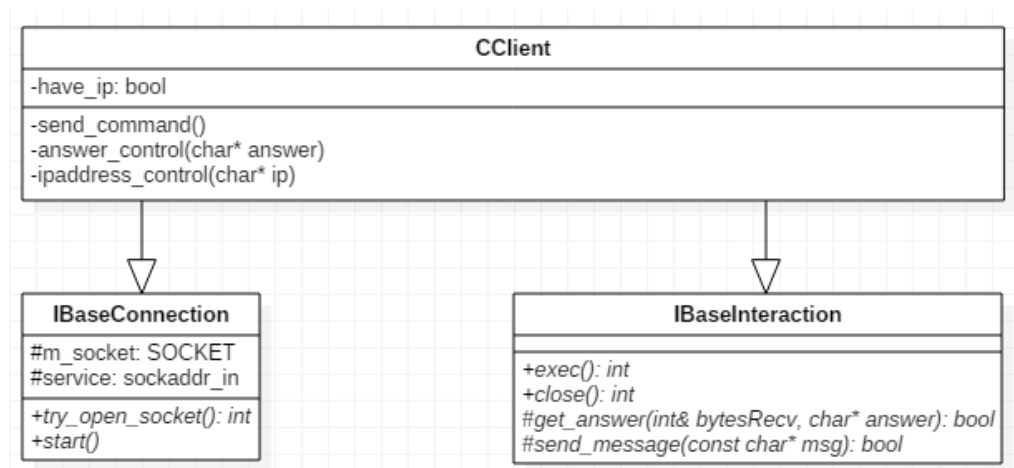
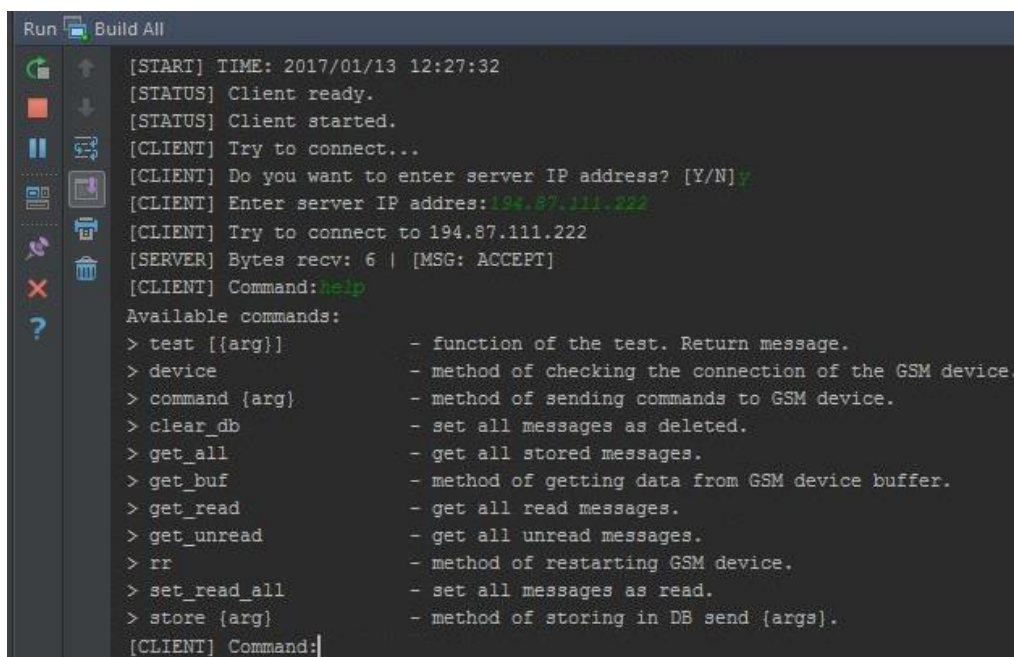


Рисунок 22 – Диаграмма классов клиента пользователя

Клиент является консольным приложением. При запуске клиента пользователю будет предложено ввести адрес сервера приложений (в случае отказа от ввода будет использован стандартный адрес в настройках – 127.0.0.1). Пользовательский клиент позволяет взаимодействовать с сервером посредством набора и отправки запросов в командной строке. В консольный вывод помещаются ответы от сервера по указанному пользователем запросу. Для работы с сервером необходимо лишь ввести ключевое слово (команду) и

отправить на сервер, в случае наличия ошибки в тексте запроса в консольном окне появится сообщение о ней.

Вид интерфейса командной строки пользователя приведен на Рис. 23.



```
Run Build All
[START] TIME: 2017/01/13 12:27:32
[STATUS] Client ready.
[STATUS] Client started.
[CLIENT] Try to connect...
[CLIENT] Do you want to enter server IP address? [Y/N] y
[CLIENT] Enter server IP address: 194.87.111.222
[CLIENT] Try to connect to 194.87.111.222
[SERVER] Bytes recv: 6 | [MSG: ACCEPT]
[CLIENT] Command: help
Available commands:
> test [{arg}]           - function of the test. Return message.
> device                 - method of checking the connection of the GSM device.
> command {arg}          - method of sending commands to GSM device.
> clear_db               - set all messages as deleted.
> get_all                 - get all stored messages.
> get_buf                 - method of getting data from GSM device buffer.
> get_read                - get all read messages.
> get_unread              - get all unread messages.
> rr                     - method of restarting GSM device.
> set_read_all            - set all messages as read.
> store {arg}             - method of storing in DB send {args}.
[CLIENT] Command:
```

Рисунок 23 – Снимок интерфейса командной строки пользователя

2.5.3.1 Описание модулей сервера приложений

Для работы клиента части приложений используются следующие модули:

- connection.h;
- interaction.h;
- client.h;
- config.h.

Исходный код программного обеспечения клиента представлен в Приложении Б.

2.5.3.1.1 Менеджер соединений

Модуль connection.h – модуль базового интерфейса менеджера соединений. Данный модуль описывает основные поля и методы для реализации сетевого взаимодействия работы с входящими подключениями к серверу. Модуль содержит в себе абстрактный класс – IBaseConnection.

2.5.3.1.2 Модуль взаимодействий

Модуль `interaction.h` – модуль базового интерфейса менеджера взаимодействий. Данный модуль описывает основные поля и методы для реализации взаимодействия с подключенными к серверу клиентами. Модуль содержит в себе абстрактный класс – `IBaseInteraction`.

2.5.3.1.3 Основной модуль

Модуль `client.h` – основной модуль клиента. Позволяет создать объект клиента. Основные поля и методы для реализации взаимодействия сетевого взаимодействия для взаимодействия с сервером приложений описаны в родительских классах `IBaseConnection` и `IBaseInteraction`. Модуль содержит в себе основной класс для работы с клиентом – `CClient`. Данный класс осуществляет работу клиента используя следующие методы:

- Функция `int try_open_socket()` - метод осуществляющий инициализацию сокета и отправку запроса на подключение клиента к серверу. Возвращает 1 если открыть сокет получилось, 0 если сокет не удалось открыть, возвращает значение -1 если запрос на подключение отклонен. Данный метод содержит в себе логику настройки сетевого подключения к серверу.
- Функция `int exec()` - основной метод класса. Возвращает 0, если работа завершилась без аварий, иначе возвращает значение -1. Данный метод содержит в себе логику взаимодействия подключенного клиента с сервером, обращаясь к менеджеру соединений для отправки запроса на подключение клиента к серверу и ожидания результата запроса, а также обращаясь к модулю взаимодействий для работы с сервером (отправка операционных кодов, отображение полученных результатов).
- Функция `void start()` – метод, позволяющий начать работу клиентской части. Данный метод осуществляет инициализацию модулей, конфигурацию сетевого взаимодействия. Метод содержит логику работы с сетевыми соединениями - позволяет начать работу с сетью, и осуществлять

подключение к серверу, передачу информации, благодаря запуску методов `CClient::try_open_socket()`, `CClient::exec()`.

- Функция `void close()` – метод, реализующий закрытие рабочего соединения с сервером. Производится завершение работы клиента.
- Функция `void answer_control(char* answer)` - метод осуществляющий контроль ввода ответа пользователя на вопрос с булевым ответом (да/нет). Параметр `answer` – ответ пользователя на вопрос. Логика работы метода заключается в организации ввода ответа пользователя и проверки ответа на корректность, исходя из допустимого множества значений ответа [«у», «п»].
- Функция `void ipaddress_control(char* ip)` – метод, осуществляющий контроль ввода ip-адреса сервера четвертой версии (ipv4). Параметр `ip` – введенный пользователем ip-адрес сервера. Логика работы метода заключается в организации ввода ip-адреса сервера пользователем и проверки ответа на соответствие синтаксису ipv4.

2.5.3.1.4 Модуль настроек

Модуль `config.h` – модуль конфигураций клиента. Данный модуль содержит основные константы, основные параметры и сетевые настройки клиентской части. Модуль содержит в себе пространство имен – `ServerCfg`, в котором расположены все необходимые атрибуты.

2.5.4 Программное обеспечение анализатора трафика

Программное обеспечение анализатора трафика обеспечивает выполнение основной цели данной выпускной квалификационной работы, а именно осуществление возможности без вмешательства в исследуемую систему отслеживать поток данных по USB каналу связи, и при необходимости изменять его, в целях исправления ошибок, отладки, или управления исследуемой системой.

В качестве ведомого устройства будет выступать клавиатура, а в качестве ведущего – персональный компьютер.

Как уже было отмечено в разделе 2.1. Анализ трафика, анализатор USB-трафика выполняет работу концентратора, осуществляя ретрансляцию входящих пакетов с одного USB-порта в другой. Для сбора и анализа трафика устройство не просто ретранслирует входящие кадры, но также и копирует кадры, и производит их обработку. Обработка скопированных пакетов заключается в том, что из кадров извлекается информация поля данных пакета и полученные данные из двоичного формата декодируются в удобный для человека вид (набор ASCII-кодов).

Результаты обработки кадров хранятся в буфере устройства. Как только размер буфера достигнет критической отметки, то незамедлительно осуществляется отправка данных на удаленный сервер. Передача данных осуществляется с использованием стека протоколов TCP/IP на удаленный сервер, вне зависимости от наличия подключения к сети Интернет исследуемого объекта, трафик которого анализируется.

Хранение данных в буфере, позволяет осуществлять удаленный контроль, путем отправки пакетов ведущему устройству, с целью корректировки передаваемых данных (исправления ошибок передачи), а также для проведения отладки исследуемой системы. Как и передача хранимых данных, управление ведущим устройством производится удаленно.

Для работы с анализатором трафика используется специализированный интерфейс клиента, благодаря которому осуществляется контроль работы анализатора при помощи управляющих сигналов.

Также под данный интерфейс определены специализированные команды, при помощи которых можно:

- осуществлять удаленный контроль путем отправки необходимых пакетов на устройство, где в роли получателя будет выступать исследуемая система;
- осуществлять удаленную конфигурацию устройства путем отправки необходимых конфигураций.

Интерфейс клиента анализатора позволяет взаимодействовать с устройством как средствами сервера, так и средствами мобильной связи, а именно используя службу коротких сообщений.

О готовности устройства к работе и о взаимодействии пользователя с устройством сообщает встроенный динамик.

Для организации логики работы анализатора трафика применяется модульное программирование. На рисунке 23 изображена структурная диаграмма, которая демонстрирует разбиение программного обеспечения на некоторые структурные компоненты, между которыми определены необходимые и соответствующие связи.

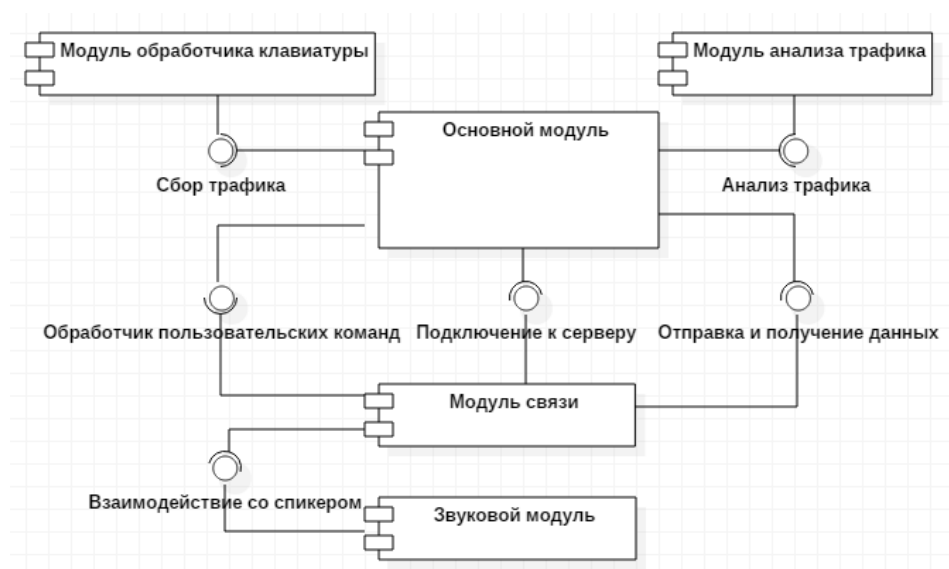


Рисунок 23 – Диаграмма компонентов программного обеспечения анализатора трафика

После определения структуры компонентов можно построить диаграмму классов, демонстрирующую классы, их атрибуты, методы и взаимосвязи между ними (Рис. 24).

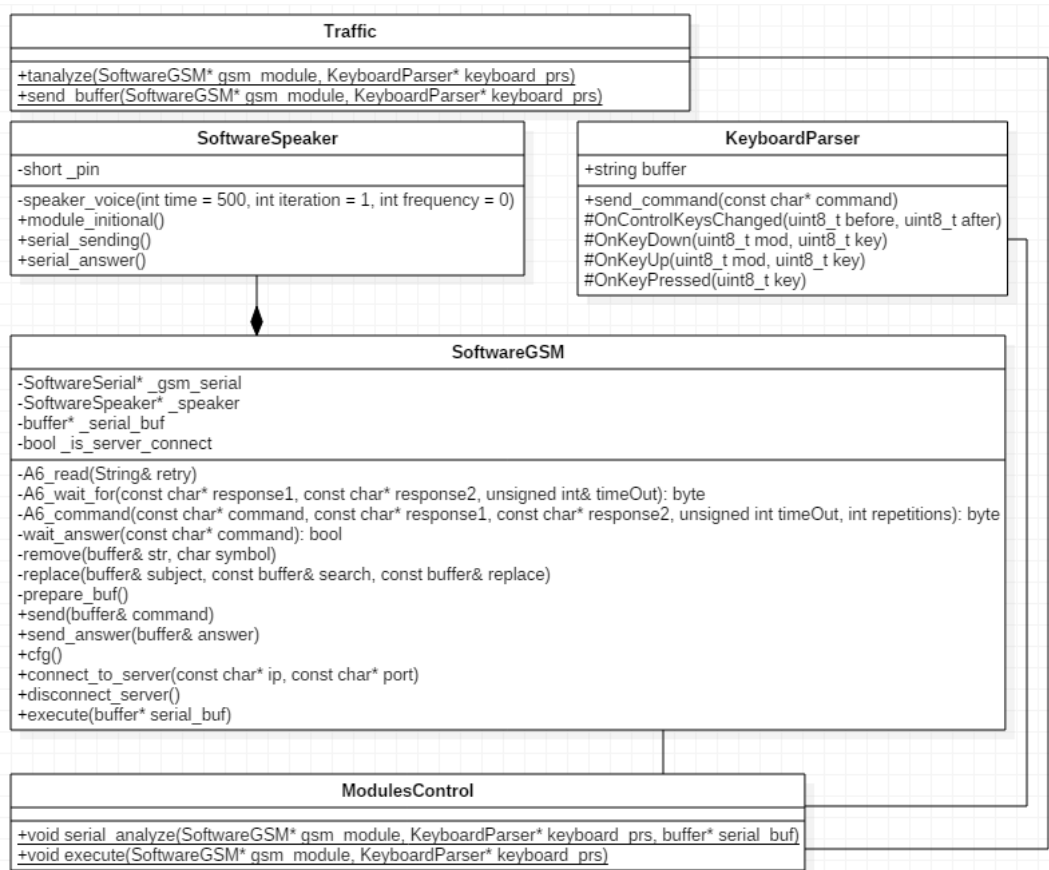


Рисунок 24 – Диаграмма классов программного обеспечения анализатора трафика

2.5.4.1 Описание модулей сервера приложений

Для работы сервера приложений используются следующие модули:

- controller.h;
- gsm.h;
- keyboard.h;
- speaker.h;
- traffic.h.

Исходный код программного обеспечения анализатора трафика представлен в Приложении А.

2.5.4.1.1 Основной модуль

Модуль `controller.h` – основной модуль анализатора трафика. Данный модуль описывает основные методы работы с трафиком и с командами пользователей, а также осуществляет запуск и инициализацию модулей связи, анализа трафика, обработчика клавиатуры, обеспечивает взаимодействия между ними. Модуль содержит в себе класс – `ModulesControl`. Данный класс осуществляет контроль работы всех объявленных модулей с использованием следующих методов:

- Функция `void serial_analyze(SoftwareGSM* gsm_module, KeyboardParser* keyboard_prs, buffer* serial_buf)` – метод осуществляющий обработку пользовательских команд полученных как от сервера, так и от службы коротких сообщений. Логика заключается в осуществлении приема и анализа входящих обращений на модуль связи. По результатам анализа выполняется та или иная команда. Список команд:
 - Команда подключения к серверу по ip-адресу четвертой версии. Вид команды: «!ip: xxx.xxx.xxx.xxx», где xxx.xxx.xxx.xxx – адрес сервера.
 - Команда перезапуска анализатора трафика. Вид команды: «!tr»;
 - Команда незамедлительного получения буфера анализатора. Вид команды: «!get».
 - Команда отправки откорректированного трафика на ведущее устройство. Вид команды: «!send: <data>», где <data> -массив, отправляемых на устройство данных.

Параметры: `gsm_module` – указатель на объект модуля связи, `keyboard_prs` указатель на объект модуля обработчика клавиатуры, `serial_buf` – указатель на буфер устройства;

- Функция `void execute(SoftwareGSM* gsm_module, KeyboardParser* keyboard_prs)` - основной метод класса. Данный метод содержит в себе логику инициализации буфера анализатора, взаимодействия со всеми

необходимыми модулями, осуществляет запуск методов по сбору и анализу трафика, метода для обработки пользовательских команд. Параметры: `gsm_module` – указатель на объект модуля связи, `keyboard_prs` указатель на объект модуля обработчика клавиатуры.

2.5.4.1.2 Модуль связи

Модуль `gsm.h` – модуль связи анализатора трафика. Данный модуль описывает основные поля и методы для реализации взаимодействия с GSM/GPRS модулем Ai-Thinker A6. Модуль содержит в себе класс – `SoftwareGSM`. Данный класс содержит логику подключения к удаленному серверу с использованием стека протоколов TCP/IP. А также содержит логику по приему, передаче и обработке данных сети и службы коротких сообщений.

Стоит отметить, что основное назначение класса заключается в поддержке взаимодействия GSM/GPRS модуля с аппаратной платформой. Взаимодействие осуществляется посредством универсального асинхронного передатчика (УААП, на англ. UART), в котором прием и передача осуществляется одновременно. Данный класс содержит логику приема и передаче данных по протоколу UART с аппаратной платформой.

Для настройки параметров модуля и взаимодействия с технологией мобильной связи GSM модулю необходимо передавать AT-команды. Наборы AT-команд являются набором коротких текстовых строк, которые понимает только GSM/GPRS модуль. Список использованных AT-команд представлен в приложении В.

Данный класс использует следующие методы:

- Функция `void send(buffer& command)` – метод передачи данных на GSM/GPRS модуль по протоколу UART. Параметр: `command` – буфер передаваемых данных.
- Функция `void send_answer(buffer& answer)` – метод передачи данных удаленному серверу. Параметр `answer` – буфер передаваемых данных.

- Функция `void cfg()` – метод передачи AT-команд, осуществляющих инициализацию и настройку GSM/GPRS модулю.
- Функция `void connect_to_server(const char* ip, const char* port)` – метод, осуществляющий подключение к удаленному серверу по указанному ip-адресу и порту. Параметр `ip` – адрес сервера четвертой версии, `port` – номер порта.
- Функция `void disconnect_server()` – метод, осуществляющий разрыв соединения с удаленным сервером.
- Функция `void execute(buffer* serial_buf)` – основной метод класса. Параметр: `serial_buf` – указатель на буфер модуля связи. Данный метод осуществляет прослушивание GSM/GPRS модуля. Полученные данные модуля помещаются в буфер модуля связи для дальнейшей их обработки.
- Функция `void A6_read(String& retry)` – метод чтения данных, отправленных с GSM/GPRS модуля на аппаратную платформу. Параметр `retry` – массив переданных аппаратной платформе данных.
- Функция `byte A6_wait_for(const char* response1, const char* response2, unsigned int& timeOut)` – метод ожидания ответа обработки переданной AT-команды на GSM/GPRS модуль. Параметры: `response1`, `response2` – ответы, ожидаемые от модуля, `timeOut` – время ожидания ответа. Возвращает 1 – если один из ожидаемых ответов получен, 2 – если получен ответ не соответствующий ожидаемым, 3 – если время ожидания превышено.
- Функция `byte A6_command(const char* command, const char* response1, const char* response2, unsigned int timeOut, int repetitions)` – метод передачи AT-команды с определенным временем ожидания ответов, и с определенным количеством попыток передачи данной команды модулю. Параметры: `response1`, `response2` – ожидаемые ответы от модуля, `timeOut` – время ожидания ответа, `repetitions` – количество передачи команды в случае каких-либо неудач, `command` – символьная строка с AT-командой. Возвращает 1 – если один из ожидаемых ответов получен, 2 – если получен

ответ не соответствующий ожидаемым, 3 – если время ожидания превышено.

- Функция `bool wait_answer(const char* command)` – метод передачи АТ-команды модулю с ожиданием успешного выполнения команды. Команда вызывающая метод `A6_command()`, ожидающая ответ «ОК», с заданным временем ожиданием 5000 мс, количеством попыток 5. Параметр `command` – символьная строка с АТ-командой. Возвращает истину, если команда выполнена успешно, или ложь, если команда выполнена не выполнена, или выполнена с ошибками.
- Функция `void remove(buffer& str, char symbol)` – метод, удаляющий в заданной строке `str` все вхождения символа `symbol`.
- Функция `void replace(buffer& subject, const buffer& search, const buffer& replace)` – метод, осуществляющий замену в строке `subject` подстроки `search` на строку `replace`.
- Функция `void SoftwareGSM::prepare_buf()` – метод, осуществляющий подготовку буфера GSM/GPRS модуля после завершения пересылки данных.

2.5.4.1.3 Модуль обработчика клавиатуры

Модуль `keyboard.h` – модуль обработчика клавиатуры. Модуль содержит в себе класс – `KeyboardParser`. Данный класс осуществляет основную задачу анализатора трафика - выполняет работу концентратора, осуществляя ретрансляцию входящих пакетов с одного USB-порта в другой, а также осуществляет сбор трафика путем копирования кадров при их ретрансляции. Полученные данные помещаются в специализированный буфер анализатора трафика `KeyboardParser.buffer`.

Данный класс осуществляет работу концентратора и сборщика трафика используя следующие методы:

- Функция `void send_command(const char* command)` – метод передачи текста, поступающего с клавиатуры ведущему устройству. Данный метод реализует передачу скорректированного набора данных ведущему устройству от ведомого. Параметр `command` – массив передаваемых данных.
- Функция `void OnControlKeysChanged(uint8_t before, uint8_t after)` – обработчик событий взаимодействия с функциональными клавишами и специальными клавишами. Параметры `before` – набор бит клавиш управления до нажатия, `after` – набор бит клавиш управления после нажатия.
- Функция `void OnKeyDown(uint8_t mod, uint8_t key)` – обработчик события нажатия какой-либо клавиши. Параметры `mod` – набор бит нажатия специальных клавиш (`ctrl`, `alt`, `shift`, `win/gui`), `key` – шестнадцатеричный код нажатой клавиши.
- Функция `void OnKeyUp(uint8_t mod, uint8_t key)` – обработчик события отжатой клавиши. Параметры `mod` – набор бит нажатия специальных клавиш (`ctrl`, `alt`, `shift`, `win/gui`), `key` – шестнадцатеричный код отжатой клавиши.
- Функция `void OnKeyPressed(uint8_t key)` – обработчик события зажатой клавиши. Параметр `key` – шестнадцатеричный код зажатой клавиши.

2.5.4.1.5 Звуковой модуль

Модуль `speaker.h` – звуковой модуль. Данный модуль содержит основные методы для взаимодействия с подключенным к аппаратной платформе динамика (спикера). Спикер используется для определения запуска и инициализации модуля связи, а также для определения данных, поступающих на модуль связи. Модуль содержит в себе класс – `SoftwareSpeaker`. Для работы со спикером используются следующие методы данного класса:

- Функция `void speaker_voice(int time, int iteration, int frequency)` – метод осуществляет подачу аналогового сигнала определенной частоты с

определенным интервалом времени и периодичностью. Параметры `time` – длительность сигнала, `iteration` – количество повторов передачи сигнала, `frequency` – частота сигнала в Гц.

- Функция `void module_initinal()` – метод воспроизведения звука при инициализации модуля связи.
- Функция `void serial_sending()`– метод воспроизведения звука при получении данных модулем связи от пользователя.
- Функция `void serial_answer()`– метод воспроизведения звука при передаче данных модулем связи удаленному серверу.

2.5.4.1.6 Модуль анализа трафика

Модуль `traffic.h` – модуль анализа трафика. Модуль содержит в себе класс – `Traffic`. Данный модуль осуществляет анализ трафика, собранного модулем обработчика клавиатуры. Работа модуля заключается в том, что из кадров извлекается информация поля данных пакета и полученные данные из двоичного формата декодируются в удобный для человека вид (набор ASCII-кодов). Обработанные данные помещаются в буфер, и, как только собирается необходимое количество данных в буфере, или, как только приходит команда о незамедлительной выгрузке буфера, данный модуль отправляет данные на удаленный сервер.

Для осуществления анализа трафика используются следующие методы:

- Функция `void tanalyze(SoftwareGSM* gsm_module, KeyboardParser* keyboard_prs)` – основной метод класса. Данный метод содержит в себе логику анализа трафика модулем обработчика клавиатуры. Параметры: `gsm_module` – указатель на объект модуля связи, `keyboard_prs` указатель на объект модуля обработчика клавиатуры.
- Функция `void send_buffer(SoftwareGSM* gsm_module, KeyboardParser* keyboard_prs)` – метод, осуществляющий передачу обработанного трафика удаленному серверу. Параметры: `gsm_module` – указатель на объект

модуля связи, keyboard_prs указатель на объект модуля обработчика клавиатуры.

2.5.5 Проектирование корпуса устройства

Для безопасной работы с устройством необходимо иметь корпус, преимущество которого также заключается в том, что использование устройства становится удобным и простым.

Модель корпуса достаточно проста и не содержит мелких деталей. Данная модель создана в графическом редакторе Blender. Эскиз модели представлен на рисунках 25-26.

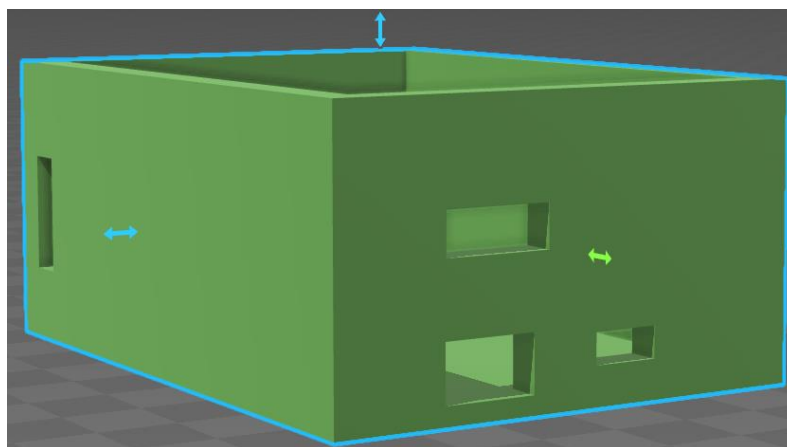


Рисунок 25 – Модель основного корпуса устройства

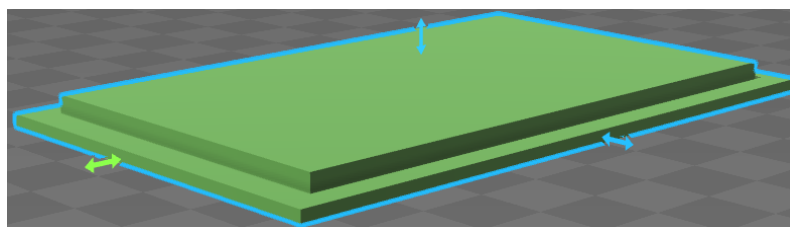


Рисунок 26 – Модель крышки корпуса устройства

Используя данную модель можно получить корпус в результате работы 3d-принтера. Корпус выполнен из SBS пластика, преимущества использования заключается в том, что данный пластик:

- дешевый;
- имеет гибкую структуру, что придает корпусу прочность;

- является безопасным, не имеет запаха.

Корпус (Рис. 27) изготовленный из пластика имеет следующие характеристики: длина – 88 мм, ширина – 64 мм, высота 40 мм, толщина стенок корпуса – 2 мм.



Рисунок 27 – Внешний вид корпуса анализатора USB трафика

На корпусе имеется два отверстия, предназначенные для подключения, ведомого и ведущего устройств. Для подключения SIM-карты необходимо вынуть крышку корпуса и установить ее в GSM/GPRS модуль (Рис. 28).

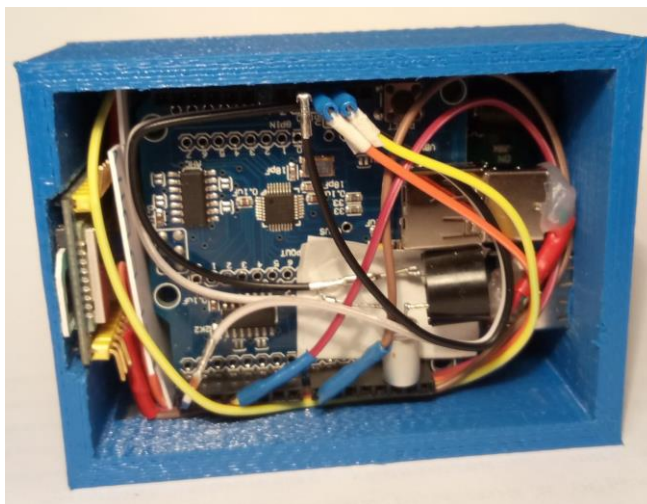


Рисунок 28 – Устройство анализатора USB трафика в корпусе

Результатом общей сборки является устройство, осуществляющее анализ трафика USB между ведущим и ведомым устройствами в пластиковом корпусе.

Устройство зафиксировано в корпусе, защищено от физических воздействий и от пыли и влаги. Данное оборудование готово к работе.

3 РЕЗУЛЬТАТЫ РАЗРАБОТКИ

3.1 Аппаратная часть

Результатом разработки аппаратной части является устройство, осуществляющее анализ USB трафика между ведущим и ведомым USB-устройствами на аппаратном уровне, а также осуществляющее взаимодействие с сервером приложений.

Устройство (Рис. 29) помещено в пластиковый корпус, который защищает компоненты устройства от физических воздействий, влаги и пыли.

Основные компоненты анализатора трафика:

- Аппаратная платформа - Arduino Leonardo.
- Плата расширения - USB Host Shield 2.0.
- GSM/GPRS модуль - Ai-Thinker A6.
- Спикер.



Рисунок 29 – Анализатор USB трафика

3.2 Программная часть

Результатом разработки программной части является программное обеспечение устройства, осуществляющее анализ трафика, а также реализация трёхуровневой архитектуры «клиент-сервер», в которой присутствуют следующие слои:

- слой клиента: пользовательский клиент, анализатор трафика;
- слой логики - сервер приложений;
- слой данных - сервер базы данных.

В результате разработки программного обеспечения анализатора трафика реализована основная цель данной выпускной квалификационной работы, а именно осуществление возможности без вмешательства в исследуемую систему отслеживать поток данных по USB каналу связи, и при необходимости изменять его, в целях исправления ошибок, отладки, или управления исследуемой системой.

Результатом разработки слоя клиента является пользовательский клиент и интерфейс пользователя, основная задача которого заключается в предоставлении пользовательского интерфейса для взаимодействия с сервером приложений и анализатором трафика. А также результатом разработки слоя клиента является интерфейс клиента анализатора трафика, позволяющий устройству осуществлять взаимодействие с сервером приложений.

Результатом разработки слоя логики является программного обеспечение, которое позволяет клиенту работать с полученным трафиком и анализатором трафика, осуществляя его контроль.

Результатом разработки слоя данных является база данных, которая осуществляет хранение и обработку полученной информации от анализатора трафика. Обработка заключается в том, что информация по запросу от сервера приложений извлекается из базы данных и отправляется в слой логики.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

4.1 Область применения

В данном разделе приведены основные сведения, необходимые для первоначальной настройки устройства и сервера и их дальнейшей эксплуатации.

Использование данного продукта позволит упростить ведение аудита для вычислительных устройств, не имеющих операционной системы с графической оболочкой, или не имеющих выход в сеть Интернет, а также позволит упростить процессы тестирования и отладки программного обеспечения, разрабатываемого для USB-устройств.

4.2 Требования к уровню подготовки

Для использования продукта пользователь должен обладать следующими навыками:

- Знание персонального компьютера.
- Навыки работы и администрирования в среде Linux на уровне опытного пользователя.

4.3 Требования к программной среде

Для работы сервера приложений и пользовательского клиента необходимо чтобы ЭВМ удовлетворяло следующим требованиям:

- ОС: UNIX-подобная операционная система, разрядность 32 или 64 бит.
- Объем ОЗУ: Минимум 128 Мб.
- Минимальный объем свободного места на жестком диске: 5 Мб.

Для работы сервера приложений необходимо наличие сервера баз данных MySQL Community Server версии 4.1 и выше.

4.4 Настройка сервера приложений

Для работы сервера необходимо, чтобы операционная система имела доступ к порту по умолчанию – 8082.

База данных по умолчанию должна быть установлена локально относительно сервера приложений. Для корректной работы сервера приложений с БД необходимо создать базу данных с нужной структурой (Код запроса на создание необходимой базы данных в Приложении Г).

Создайте папку в корне с именем «server», расположите в данной папке исходный код сервера приложений (Приложение В, Д). Для начала работы сервера необходимо выполнить сборку сервера для текущей операционной системы выполнив соответствующий bash-скрипт, указанный в Приложении Ж.

4.5 Настройка пользовательского клиента

Создайте папку в корне с именем «client», расположите там исходный код пользовательского клиента (Приложение Б, Е). Для начала работы клиента необходимо выполнить сборку клиента для текущей операционной системы, выполнив соответствующий bash-скрипт, указанный в Приложении З.

4.6 Настройка анализатора трафика

Для работы с анализатором трафика откройте крышку устройства и установите SIM-карту, у которой имеется возможность для выхода в сеть Интернет, в GSM/GPRS модуль (Рис. 30).



а – слот для SIM-карты формата microSIM;

б – SIM-карта формата microSIM

Рисунок 30 – Расположение SIM-карты устройства

После установки SIM-карты подключите ведущее и ведомое устройство в соответствующие гнезда USB (Рис. 31).



а – гнездо USB для подключения ведомого устройства;

б – окно световых индикаторов;

в – гнездо USB для подключения ведущего устройства

Рисунок 31 – Лицевая панель анализатора трафика

Следующий шаг после установки SIM-карты - дождитесь включения устройства. О полном включении устройства будет сигнализировать наличие двух звуковых сигналов длительностью 0.5 секунд:

- Первый звуковой сигнал информирует о инициализации все модулей и о переходе к регистрации в сотовой сети оператора.
- Второй звуковой сигнал информирует о том, что регистрация в сети оператора выполнена и устройство готово к использованию.

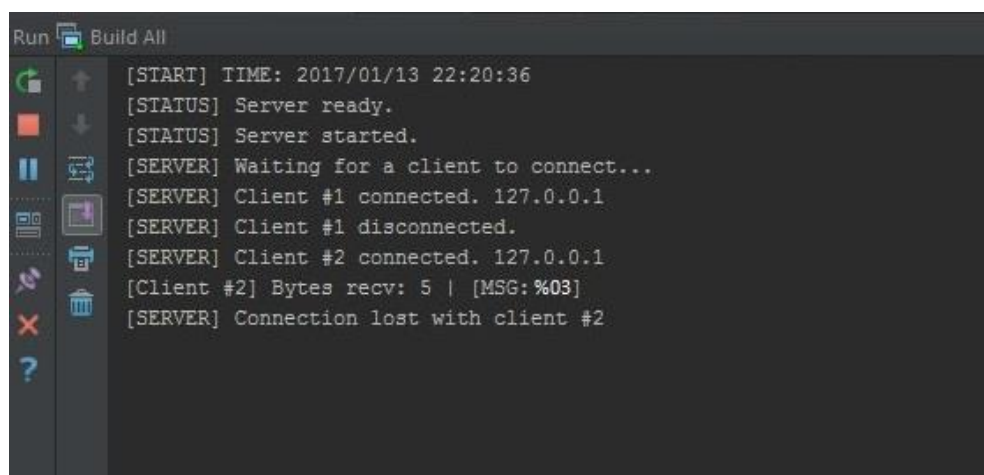
Максимальное время между первым и вторым звуковым сигналом около 1 минуты. Если затрачено больше времени, то это говорит о низком качестве связи.

Последний этап настройки устройства заключается в отправке команды для подключения к серверу приложений («!ip: xxx.xxx.xxx.xxx», где xxx.xxx.xxx.xxx – адрес сервера) посредством службы коротких сообщений на номер установленной в устройстве SIM-карты.

4.8 Описание операций сервера приложений

Сервер приложений является консольным приложением. В консольный вывод сервером будет помещаться служебная информация о входящих подключениях, о полученных командах от различных пользователей, а также о статусе сервера. Данная возможность позволяет вести аудит всех событий, происходящих на сервере приложений, отследить и выявить изменения в системе. Пример аудита приведен на Рис. 33.

Для взаимодействия с сервером необходимо использовать пользовательский клиент.



```
[START] TIME: 2017/01/13 22:20:36
[STATUS] Server ready.
[STATUS] Server started.
[SERVER] Waiting for a client to connect...
[SERVER] Client #1 connected. 127.0.0.1
[SERVER] Client #1 disconnected.
[SERVER] Client #2 connected. 127.0.0.1
[Client #2] Bytes recv: 5 | [MSG:%03]
[SERVER] Connection lost with client #2
```

Рисунок 33 – Снимок интерфейса командной строки сервера приложения.

4.9 Описание пользовательских операций анализатора трафика

Пользовательские операции для работы с устройством посредством службы коротких сообщений, необходимые для прямого взаимодействия с устройством, осуществляющим анализ USB трафика.

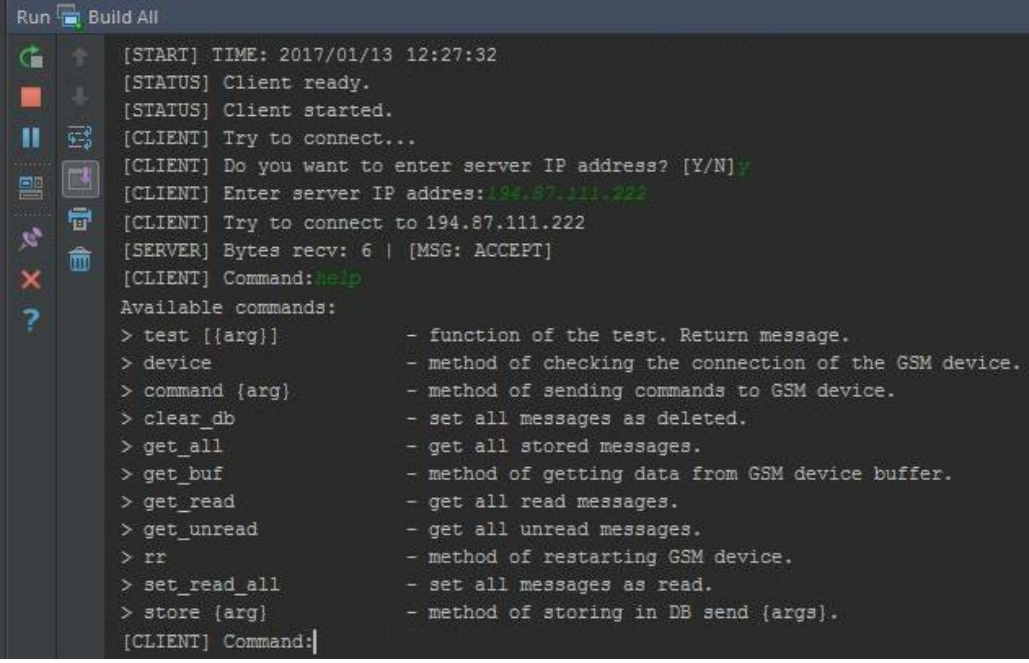
Список доступных СМС-команд:

- 1) Команда подключения к серверу по ip-адресу четвертой версии. Вид команды: «!ip: xxx.xxx.xxx.xxx», где xxx.xxx.xxx.xxx – адрес сервера.
- 2) Команда перезапуска анализатора трафика. Вид команды: «!tr».
- 3) Команда незамедлительного получения буфера анализатора. Вид команды: «!get».
- 4) Команда отправки откорректированного трафика на ведущее устройство. Вид команды: «!send: <data>», где <data> - отправляемый на устройство данные.

Для передачи необходимой команды посредством службы коротких сообщений требуется указать номер установленной в устройстве SIM-карты. Стоит отметить, что получение короткого сообщения устройством идентифицируется коротким звуковым сигналом.

4.10 Описание операций пользовательского клиента

Клиент является консольным приложением (Рис. 32). При запуске клиента пользователю будет предложено ввести адрес сервера приложений (в случае отказа от ввода будет использован стандартный адрес в настройках – 127.0.0.1).



```
[START] TIME: 2017/01/13 12:27:32
[STATUS] Client ready.
[STATUS] Client started.
[CLIENT] Try to connect...
[CLIENT] Do you want to enter server IP address? [Y/N] y
[CLIENT] Enter server IP address: 194.87.111.222
[CLIENT] Try to connect to 194.87.111.222
[SERVER] Bytes recv: 6 | [MSG: ACCEPT]
[CLIENT] Command: help
Available commands:
> test [{arg}]           - function of the test. Return message.
> device                 - method of checking the connection of the GSM device.
> command {arg}         - method of sending commands to GSM device.
> clear_db              - set all messages as deleted.
> get_all                - get all stored messages.
> get_buf               - method of getting data from GSM device buffer.
> get_read              - get all read messages.
> get_unread            - get all unread messages.
> rr                    - method of restarting GSM device.
> set_read_all           - set all messages as read.
> store {arg}           - method of storing in DB send {args}.
[CLIENT] Command:
```

Рисунок 32 – Снимок интерфейса командной строки пользователя.

Для работы с сервером необходимо лишь ввести и отправить ключевое слово (команду), в случае каких-либо ошибок в консольном окне появится соответствующая запись.

Список доступных команд:

- Команда «help» – возвращает список доступных команд с их описанием. Для выполнения данной команды наличие параметров необязательно.
- Команда «device» – возвращает информацию о подключенном устройстве анализа трафика. Для выполнения данной команды наличие параметров необязательно.
- Команда «command {arg}» – отправляет управляющие воздействия на анализатор трафика. В ней обязательно наличие аргумента arg - вектор байт.

- Команда «clear_db» – осуществляет очистку базы данных. Очистка заключается в том, что всем выбранным записям таблицы Логгер присваивается признак удаления. Для выполнения данной команды наличие параметров необязательно.
- Команда «get_all» - позволяет получить все существующие записи в базе данных, несмотря на значение флага состояния. После выполнения команды флаги состояния записей не изменяются. Для выполнения данной команды наличие параметров необязательно.
- Команда «get_buf» - позволяет в срочном порядке получить текущий буфер USB-трафика подключенного к серверу устройства. Для выполнения данной команды наличие параметров необязательно.
- Команда «get_read» - позволяет получить все существующие записи в базе данных, которые не имеют флаг состояния «Запись не прочитана» и не имеют флаг «Запись удалена». После выполнения команды флаги состояния записей не изменяются. Для выполнения данной команды наличие параметров необязательно.
- Команда «get_unread» - позволяет получить все существующие записи в базе данных, которые имеют флаг состояния «Запись не прочитана» и не имеют флаг «Запись удалена». После выполнения команды для записей выставляется флаг состояния «Запись прочитана». Для выполнения данной команды наличие параметров необязательно.
- Команда «tr» – отправляет управляющее воздействие направленное на перезапуск анализатор трафика. Для выполнения данной команды наличие параметров необязательно.
- Команда «set_read_all» - помечает все записи в базе данных флагом состояния «Запись прочитана». Для выполнения данной команды наличие параметров необязательно.

- Команда «store {arg}» - отправляет набор данных arg на хранение в базу данных. Данная команда доступна только для анализатора трафика. Для её выполнения наличие параметров обязательно.

5 ЗАКЛЮЧЕНИЕ

Целью данной выпускной квалификационной работы является осуществление возможности без вмешательства в исследуемую систему отслеживать поток данных по USB каналу связи, и при необходимости изменять его, в целях исправления ошибок, отладки, или управления исследуемой системой.

В качестве ведомого устройства выступает клавиатура, а в качестве ведущего – персональный компьютер.

Для достижения поставленной цели был обозначен ряд основных задач. Для решения всех задач была разработана система, сетевая архитектура которой представляет из себя трёхуровневую модель «клиент-сервер».

Для решения задачи сбора USB трафика, проходящего по каналу связи между ведущим и ведомым USB-устройствами, было разработано устройство на базе аппаратной платформы Arduino Leonardo, использующее для сетевого взаимодействия GSM/GPRS модуль Ai-Thinker A6. Анализатор трафика помещен в пластиковый корпус, который защищает его от физических воздействий, а также от пыли и влаги. Использование данного устройства позволяет осуществить, без вмешательства в исследуемую систему, наблюдение за потоком данных, проходящим по USB каналу связи между ведущим и ведомым устройствами, и при необходимости изменять его, в целях исправления ошибок, отладки, или управления исследуемой системой. В качестве ведомого устройства будет выступать клавиатура, а в качестве ведущего – персональный компьютер.

Используя GSM/GPRS модуль Ai-Thinker A6 в устройстве, решается ряд задач, связанный с организацией сетевого взаимодействия анализатора с сервером приложений. Данный модуль позволяет устройству осуществлять передачу данных без взаимодействия с исследуемой системой.

В данной выпускной квалификационной для решения задачи хранения данных используется сервер базы данных, содержащий СУБД MySQL компании Oracle, распространяемая под лицензией свободного программного обеспечения GNU General Public License. Сервер базы данных выполняет обслуживание и управление базой данных и отвечает за целостность и сохранность данных, а также обеспечивает операции ввода-вывода при доступе к информации.

Для решения задач обработки USB трафика, отправки обработанных данных, обработки пользовательских команд, а также задачи работы с множеством клиентов одновременно, разработан сервер приложений.

Сервер приложений, осуществляет обработку полученных команд от клиентов, предоставляя возможность взаимодействовать с хранимым трафиком. USB трафик, предоставляемый анализатором, отправляется сервером приложений на сервер базы данных.

Взаимодействие с функционалом сервера приложений доступно пользователю в виде специализированных команд, передача на сервер которых осуществляется при помощи клиентского приложения.

Используя менеджер взаимодействий, сервер приложений может работать сразу с несколькими подключенными клиентами в режиме многопоточности.

Для решения задач стороны клиента разработано:

- Интерфейс пользователя, позволяющий отправлять на сервер необходимые запросы и выводить результаты запросов в понятном для пользователя, виде. Запросы пользователя представлены в виде команд с параметрами, которые отправляются в виде специализированных операционных кодов, которые понятны только серверу приложений.
- Интерфейс анализатора трафика, который позволяет осуществлять обмен сообщениями, управляющими сигналами между сервером приложений и устройством. С использованием этого интерфейса осуществляется контроль работы анализатора.

Интерфейс пользователя используется пользовательским клиентом. Клиент является консольным приложением. Данный клиент позволяет взаимодействовать с сервером посредством командной строки, предоставляя пользователю отправлять команды. В консольный вывод помещаются ответы от сервера по указанному пользователем запросу. Для работы с сервером необходимо лишь ввести ключевое слово (команду) и отправить на сервер, в случае каких-либо ошибок в консольном окне появится соответствующая запись.

Для решения задачи удаленного контроля устройства, осуществляющего анализ трафика, организована возможность взаимодействия с анализатором используя сервер приложений, а также службу коротких сообщений мобильной связи. Управление осуществляется с применением интерфейса анализатора трафика. Использование сервера приложений и службы коротких сообщений позволяют осуществлять контроль подключенного анализатора USB трафика.

Таким образом, задачи решены в полном объеме, цель достигнута – реализовано наблюдение за потоком данных, проходящим по каналу связи USB между ведущим и ведомым USB-устройствами на аппаратном уровне, а также возможность производить умышленное изменение трафика в целях исправления ошибок, проведения отладки, или управления исследуемой системой. Достигается возможность проведения исследования ведущего устройства без вмешательства непосредственно в ее работу с ведомым устройством, что также позволяет значительно расширить круг анализируемых систем.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Агуров, П.В., Интерфейс USB. Практика использования и программирования [Текст]: учеб. пособие / П.В. Агуров. – СПб.: Изд-во БХВ-Петербург, 2006. – 576 с.
2. Гук, М., Аппаратные средства IBM PC [Текст]: учеб. пособие / М. Гук. - СПб: Питер, 2000. - 1072 с.
3. Хульцебош Ю. USB в электронике [Текст]: учеб. пособие / Хульцебош Ю. – СПб.: Изд-во ЭКСМО, 2011. – 224 с.
4. Кулаков, В., Программирование на аппаратном уровне: специальный справочник [Текст]: монография. – СПб.: Изд-во Питер, 2003. – 847 с.
5. Многоуровневые системы клиент-сервер. [Электронный ресурс] URL: <https://www.osp.ru/nets/1997/06/142618/> (09.02.2017).
6. Закиров, З.Г., Сотовая связь стандарта GSM. Современное состояние, переход к сетям третьего поколения [Текст]: учеб. пособие / З.Г. Закиров, А.Ф. Надеев, Р.Р. Файзуллин. - М.: Эко-Трендз, 2004. – 264 с.
7. Степурин, А.В., Протокол GPRS. Краткие инструкции для новичков [Текст]: монография. – М.: Изд-во ВЯТКА, 2005. – 128 с.
8. Попов В.И. Основы сотовой связи стандарта GSM [Текст]: учеб. пособие / В.И. Попов. – М.: Изд-во Эко-Трендз, 2005. – 295 с.
9. STM32VLDISCOVERY. [Электронный ресурс] URL: <http://www.st.com/en/evaluation-tools/stm32vldiscovery.html> (10.02.2017)
- 10.Руководство пользователя STM32VLDISCOVERY. [Электронный ресурс] URL: http://www.st.com/content/ccc/resource/technical/document/user_manual/f3/16/fb/63/d6/3d/45/aa/CD00267113.pdf/files/CD00267113.pdf/jcr:content/translations/en.CD00267113.pdf (10.02.2017).
- 11.Руководство пользователя Raspberry Pi. [Электронный ресурс] URL: <http://www.cs.unca.edu/~bruce/Fall14/360/RPiUsersGuide.pdf> (10.02.2017).

12. RASPBERRY PI 3 MODEL B. [Электронный ресурс] URL:
<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/> (10.02.2017)
13. Arduino Leonardo. [Электронный ресурс] URL:
<https://www.arduino.cc/en/main/arduinoBoardLeonardo> (10.02.2017)
14. Платы расширения Arduino. [Электронный ресурс] URL:
<https://www.arduino.cc/en/Main/arduinoShields> (10.02.2017)
15. Иго, Т., Arduino, датчики и сети для связи устройств [Текст]: учеб. для вузов / Т. Иго. – СПб.: Изд-во БХВ-Петербург, 2015. – 544 с.
16. Техническая спецификация SIM800L. [Электронный ресурс] URL:
http://wiki.seeedstudio.com/images/4/46/SIM800L_Hardware_Design_V1.00.pdf (12.02.2017).
17. Руководство пользователя A6/A7/A6C. [Электронный ресурс] URL:
http://wiki.seeedstudio.com/images/4/46/SIM800L_Hardware_Design_V1.00.pdf (12.02.2017).
18. Техническая спецификация USB HOST SHIELD 2.0 [Электронный ресурс] URL:
<http://tinkbox.ph/sites/tinkbox.ph/files/downloads/USB%20HOST%20SHIELD.pdf> (12.02.2017).
19. Васвани, В., MySQL: использование и администрирование [Текст]: учеб. пособие / В. Васвани. — М.: Изд-во Питер, 2011. — 368 с.
20. Гольцман, В.И., MySQL 5.0. Библиотека программиста [Текст]: учеб. пособие / В.И. Гольцман. — М.: Изд-во Питер, 2009. — 253 с.

ПРИЛОЖЕНИЕ А

```
//
```

```
=====
```

```
=====
```

```
#ifndef TRAFFIC_H
```

```
#define TRAFFIC_H
```

```
//
```

```
=====
```

```
=====
```

```
#include "../includes/device/gsm/gsm.h"
```

```
#include "../includes/device/keyboard/keyboard.h"
```

```
//
```

```
=====
```

```
=====
```

```
const uint8_t MAX_BUFFER_SIZE = 10;
```

```
class Traffic {
```

```
public:
```

```
    static void tanalyze(
```

```
        SoftwareGSM* gsm_module,
```

```
        KeyboardParser* keyboard_prs
```

```
    );
```

```
    static void send_buffer(
```

```
        SoftwareGSM* gsm_module,
```

```
        KeyboardParser* keyboard_prs
```

```

    );

};

//
=====

=====

#endif /* TRAFFIC_H */

//
=====

=====

//
=====

=====

#ifndef SPEAKER_H

#define SPEAKER_H

//
=====

=====

#include "Arduino.h"

//
=====

=====

class SoftwareSpeaker

{

public:

    SoftwareSpeaker(short pin = 0);

    void module_inital();

```

```

        // void sms_sending();

        void serial_sending();

        void serial_answer();

private:

        short _pin;

        void speaker_voice(int time = 500, int iteration = 1, int frequency = 0);

};

//
=====

=====

#endif /* SPEAKER_H */

//
=====

=====

//
=====

=====

#ifndef MOUSE_H

#define MOUSE_H

//
=====

=====

#include "Mouse.h"

#include <hidboot.h>

#include <usbhub.h>

```

```
//
```

```
=====
```

```
=====
```

```
class MouseParser : public MouseReportParser {
```

```
protected:
```

```
    void OnMouseMove(MOUSEINFO *mi);
```

```
    void OnLeftButtonUp(MOUSEINFO *mi);
```

```
    void OnLeftButtonDown(MOUSEINFO *mi);
```

```
    void OnRightButtonUp(MOUSEINFO *mi);
```

```
    void OnRightButtonDown(MOUSEINFO *mi);
```

```
    void OnMiddleButtonUp(MOUSEINFO *mi);
```

```
    void OnMiddleButtonDown(MOUSEINFO *mi);
```

```
};
```

```
=====
```

```
=====
```

```
#endif /* MOUSE_H */
```

```
//
```

```
=====
```

```
=====
```

```
/*
```

```
=====
```

```
===== */
```

```
#ifndef KEYBOARD_H
```

```
#define KEYBOARD_H
```

```

/*
=====

===== */

#include <Keyboard.h>

#include <hidboot.h>

#include <usbhub.h>

#include <ArduinoSTL.h>

/*
=====

===== */

class KeyboardParser : public KeyboardReportParser
{
public:
    std::string buffer;

    void send_command(const char* command);

protected:
    void OnControlKeysChanged(uint8_t before, uint8_t after);

    void OnKeyDown(uint8_t mod, uint8_t key);

    void OnKeyUp(uint8_t mod, uint8_t key);

    void OnKeyPressed(uint8_t key);

};

/*
=====

===== */

#endif /* KEYBOARD_H */

```

```

/*
=====

===== */

//

=====

=====

#ifndef GSM_H

#define GSM_H

//

=====

=====

#include "Arduino.h"

#include "../includes/device/speaker/speaker.h"

#include <ArduinoSTL.h>

#include <avr/pgmspace.h>

#include <SoftwareSerial.h>

//

=====

=====

#define DEFAULT_RX 8

#define DEFAULT_TX 3

#define DEFUALT_POWER_PIN 2

#define DEFAULT_SERIAL_PORT 9600

#define OK 1

#define NOT_OK 2

```

```
#define TIMEOUT 3
```

```
//
```

```
=====
```

```
=====
```

```
void viewFreeMemory();
```

```
//
```

```
=====
```

```
=====
```

```
#define FF(str) String(str).c_str()
```

```
typedef std::string buffer;
```

```
//
```

```
=====
```

```
=====
```

```
class SoftwareGSM
```

```
{
```

```
public:
```

```
    SoftwareSerial *_gsm_serial;
```

```
    SoftwareGSM(
```

```
        const short& rx = DEFAULT_RX,
```

```
        const short& tx = DEFAULT_TX,
```

```
        const long& serial_port = DEFAULT_SERIAL_PORT
```

```
    );
```

```
    void send(buffer& command);
```

```
    void send_answer(buffer& answer);
```

```
    // void send_sms(const char* phone_number, const char* text);
```

```

// void call_number(const char* phone_number);

void cfg();

void connect_to_server(const char* ip, const char* port);

void disconnect_server();


void execute(buffer* serial_buf);

private:

SoftwareSpeaker* _speaker;

buffer* _serial_buf;

bool _is_server_connect;


void A6_read(String& retry);

byte A6_wait_for(

    const char* response1,

    const char* response2,

    unsigned int& timeOut

);

byte A6_command(

    const char* command,

    const char* response1,

    const char* response2,

    unsigned int timeOut,

    int repetitions

);

```



```

bool wait_answer(const char* command);

void remove(buffer& str, char symbol);

void replace(
    buffer& subject,
    const buffer& search,
    const buffer& replace
);

void prepare_buf();

};

//
=====

=====

#endif /* GSM_H */

//
=====

=====

//
=====

=====

#ifndef CONTROLLER_H

#define CONTROLLER_H

//
=====

=====

```

```

#include "../includes/traffic/traffic.h"

//
=====

=====

#define MSG_FLAG "+CIEV: \"MESSAGE\""

#define RECEIVE_FLAG "+CIPRCV:"

#define IP_MSG_FLAG "!ip: "

#define RR_MSG_FLAG "!rr"

#define GET_MSG_FLAG "!get"

#define SEND_MSG_FLAG "!send: "

#define NPOSE std::string::npos

//
=====

=====

class ModulesControl {
public:

    static void serial_analyze(
        SoftwareGSM* gsm_module,
        KeyboardParser* keyboard_prs,
        buffer* serial_buf
    );

    static void execute(
        SoftwareGSM* gsm_module,
        KeyboardParser* keyboard_prs
    );

```

```

};

//
=====

=====

#endif /* CONTROLLER_H */

//
=====

=====

//
=====

=====

#include "../includes/core/controller.h"

//
=====

=====

SoftwareGSM *gsm_module;

//
=====

=====

USB usb;

HIDBoot<USB_HID_PROTOCOL_KEYBOARD> hid_keyboard(&usb);

KeyboardParser keyboard_prs;

//
=====

=====

void setup() {

```

```

Serial.begin(DEFAULT_SERIAL_PORT);

digitalWrite(DEFAULT_POWER_PIN, HIGH);


Keyboard.begin();

if (usb.Init() == -1) Serial.println(F("[ERROR] OSC did not start.));

hid_keyboard.SetReportParser(0, &keyboard_prs);


while(!Serial);


gsm_module = new SoftwareGSM();

// gsm->connect_to_server(FF(F("31.207.78.188")), FF(F("8082")));


Serial.println(F("[GSM] module started.));

viewFreeMemory();


keyboard_prs.buffer = "";

}

//

=====

=====

void loop() {

    ModulesControl::execute(gsm_module, &keyboard_prs);

    usb.Task();

}

```

//

=====

=====

ПРИЛОЖЕНИЕ Б

```
/*
=====
=====

* File: client.h

* Description: Реализация работы пользователя с сервером

* Created: 01.02.2016

* Author: soltanoff

*
=====
===== */

#ifndef CLIENT_H

#define CLIENT_H

/*
=====
===== */

#include "../core/interaction.h"

#include "../core/connection.h"

/*
=====
===== */

/*!

* @class CClient

* @inherit IBaseInteraction

* @inherit IBaseConnection
```

* Класс подключаемого клиента к серверу.

*/

```
class CClient: public IBaseInteraction, public IBaseConnection {
```

```
public:
```

```
    CClient();
```

```
    ~CClient();
```

```
    /*!
```

* @public Метод осуществляющий инициализацию сокета и дальнейшего подключения к серверу по введенному адресу.

* @return 0 если открыть сокет получилось, иначе -1

* @note Содержит в себе логику подключения к серверу по указанному адресу

```
    */
```

```
    int try_open_socket();
```

```
    /*!
```

* @public Основной циклический метод класса.

* @return 0 если работа завершилась без аварий, иначе -1

* @note основной метод для взаимодействия с сервером

```
    */
```

```
    int exec();
```

```
    /*!
```

* @public Метод позволяющий начать работу клиентской части.

* @return None

* @note запуск методов CClient::try_open_socket(), CClient::exec()

```

*/

void start();

/*!

* @public Метод закрывающий сокет клиента.

* @return None

* @note закрытие сокета с параметром SHUT_WR

*/

void close();

private:

    //! @private флаг указывающий был ли указан ip-адрес сервера

    bool have_ip;

    /*!

    * @private Метод осуществляющий обработку введенных команд
пользователя

    * @return None

    * @note осуществляет отправку обработанных данных пользователя на
сервер

    */

    void send_command();

    /*!

    * @private Метод отправки данных на сервер

    * @param msg - массив данных отправляемых на сервер

    * @return true - данные отправлены успешно, иначе false

```


* @note осуществляет отправку обработанных данных пользователя на сервер

*/

bool send_message(const char* msg);

/*!

* @private Метод обработки ответа от сервера

* @param bytesRecv - количество полученных байт от сервера (out)

* @param answer - массив данных, содержащий ответ от сервера (out)

* @return true - данные получены успешно, иначе false

* @note осуществляет обработку полученного ответа сервера

*/

bool get_answer(int& bytesRecv, char* answer);

/*!

* @private Метод осуществляющий контроль ввода ответа пользователя на булевый вопрос

* @param answer - ответ пользователя на вопрос [Y,N]

* @return None

* @note организовывается ввод ответа пользователя и дальнейшая его проверка ответа на корректность

*/

void answer_control(char* answer);

/*!

* @private Метод осуществляющий контроль ввода ip-адреса сервера

* @param ip - корректно введенный пользователем ip-адрес (out)

```

* @return None

* @note организовывается ввод ip-адреса и дальнейшая его проверка на
корректность

*/

void ipaddress_control(char* ip);

};

#endif /* CLIENT_CLIENT_H */

/*
=====
=====

* File: config.h

* Description: Файл основных конфигураций сервера

* Created: 03.02.2017

* Author: soltanoff

*
=====
===== */

#ifndef CONFIG_H

#define CONFIG_H

/*
=====
===== */

#ifdef _WIN32

#include <winsock.h>

#else

```

```

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#endif

#include <vector>

#include <time.h>

#include <string>

#include <iomanip>

#include <iostream>

#include <sys/types.h>

#include <thread>

#include <mutex>

/*
=====
===== */

#ifdef _WIN32

#define MSG_DONTWAIT 0

#else

#define SOCKET int

#define SOCKET_ERROR -1

#endif

//! основной мьютекс, синхронизирующий ввод и вывод данных в консоль
сервера

extern std::mutex MAIN_MUTEX;

```

```

/ * !

* @namespace ServerCfg

* Основные константы конфигурации сервера

*/

namespace ServerCfg {

    //! локальный ip-адрес сервера

    const char LOCALHOST[] = "127.0.0.1";

    //! используемый порт сервера

    const std::uint16_t PORT = 8082;

    //! тип протокола сервера

    const std::uint16_t PROTOCOL = IPPROTO_TCP;

    //! флаг ошибки открытия сокета (ошибка прослушивания)

    const std::uint16_t BACKLOG = 1;

    //! размер буфера отправляемых и получаемых данных

    const std::uint16_t BUFF_SIZE = 4056;

    //! @details основные параметры базы данных сервера

    //! ip-адрес сервера

    const std::string DB_ADDRESS = "127.0.0.1";

    //! имя используемой базы данных

    const std::string DATABASE = "logger";

    //! логин пользователя БД

    const std::string DB_USER = "root";

    //! пароль пользователя БД

    const std::string DB_PASSWORD = "dbpassword";

```

```
// const unsigned int port = 3306;

}

/*
=====
===== */

#endif /* CONFIG_H */
```

ПРИЛОЖЕНИЕ В

```
/*
```

```
=====
```

```
=====
```

```
* File: main.cpp
```

```
* Description: Основная функция проекта
```

```
* Created: 01.02.2017
```

```
* Author: soltanoff
```

```
*
```

```
=====
```

```
===== */
```

```
#ifdef _WIN32
```

```
#include "src/client/client.h"
```

```
#else
```

```
#include "src/server/server.h"
```

```
#include "src/client/client.h"
```

```
#endif
```

```
int main(int argc, char *argv[]) {
```

```
    time_t t = time(NULL);
```

```
    tm *aTm = localtime(&t);
```

```
    std::cout
```

```
        << "[START] TIME: "
```

```

    << aTm->tm_year + 1900 << "/" << std::setfill('0') << std::setw(2)

    << aTm->tm_mon + 1 << "/" << std::setfill('0') << std::setw(2)

    << aTm->tm_mday << " " << std::setfill('0') << std::setw(2)

    << aTm->tm_hour << ":" << std::setfill('0') << std::setw(2)

    << aTm->tm_min << ":" << std::setfill('0') << std::setw(2)

    << aTm->tm_sec << std::endl;

#ifdef _WIN32

    CClient s;

    s.start();

    system("pause");

#else

    CServer s;

    // CClient s;

    s.start();

#endif

    return 0;

}

/*
=====

=====

* File: config.h

* Description: Файл основных конфигураций сервера

* Created: 03.02.2017

* Author: soltanoff

```

```

*
=====

===== */

#ifndef CONFIG_H

#define CONFIG_H

/*
=====

===== */

#ifdef _WIN32

#include <winsock.h>

#else

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#endif

#include <vector>

#include <time.h>

#include <string>

#include <iomanip>

#include <iostream>

#include <sys/types.h>

#include <thread>

#include <mutex>

```



```

/*
=====
===== */

#ifdef _WIN32

#define MSG_DONTWAIT 0

#else

#define SOCKET int

#define SOCKET_ERROR -1

#endif

//! основной мьютекс, синхронизирующий ввод и вывод данных в консоль
сервера

extern std::mutex MAIN_MUTEX;

/*!

* @namespace ServerCfg

* Основные константы конфигурации сервера

*/

namespace ServerCfg {

    //! локальный ip-адрес сервера

    const char LOCALHOST[] = "127.0.0.1";

    //! используемый порт сервера

    const std::uint16_t PORT = 8082;

    //! тип протокола сервера

    const std::uint16_t PROTOCOL = IPPROTO_TCP;

    //! флаг ошибки открытия сокета (ошибка прослушивания)

```

```

const std::uint16_t BACKLOG = 1;

//! размер буфера отправляемых и получаемых данных

const std::uint16_t BUFF_SIZE = 4056;

//! @details основные параметры базы данных сервера

//! ip-адрес сервера

const std::string DB_ADDRESS = "127.0.0.1";

//! имя используемой базы данных

const std::string DATABASE = "logger";

//! логин пользователя БД

const std::string DB_USER = "root";

//! пароль пользователя БД

const std::string DB_PASSWORD = "dbpassword";

// const unsigned int port = 3306;

}

/*
=====
===== */

#endif /* CONFIG_H */

/*
=====
=====

* File: functions.h

* Description: Реализации функций обработки команд

* Created: 14.02.2017

* Author: soltanoff

```

```

*
=====

===== */

#ifndef SERVER_FUNCTIONS_H

#define SERVER_FUNCTIONS_H

/*
=====

===== */

#include "../database/database.h"

#include "../config/config.h"

#include <memory>

/*
=====

===== */

#define RR_MSG_FLAG "!rr"

#define GET_MSG_FLAG "!get"

#define SEND_MSG_FLAG "!send: "

/*!

* @class Functions

* Класс описывающий основные реализации обработчиков команд

*/

class Functions

{

public:

```

```

//
=====

//! @typedef вектор обработанных данных
typedef std::vector<std::uint8_t> DataVector;

// typedef std::function<void(DataVector)> FuncArg;

//! @typedef тип полиморфной оболочки функции
typedef std::function<void(DataVector, SOCKET)> FuncArg;

//
=====

/*!

* @public Метод реализующий обработку `неизвестной` команды

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку
данных, например)

* @param s - сокет объекта назначения

* @return None

* @note реализует обработку команды заглушки, при неопределенной
команды

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: -

*/

void unknown(DataVector arg, FuncArg result_cb, SOCKET s);

/*!

```

* @public Метод реализующий обработку команды перезагрузки GSM-устройства

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку данных, например)

* @param s - сокет объекта назначения

* @return None

* @note реализует обработку команды перезагрузки путем отправки управляющих сигналов на устройство по сети

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: rr

*/

void rr(DataVector arg, FuncArg result_cb, SOCKET s);

/*!

* @public Метод реализующий обработку команды запроса буфера GSM-устройства

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку данных, например)

* @param s - сокет объекта назначения

* @return None

* @note реализует обработку команды незамедлительного получения буфера USB-трафика GSM-устройства

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: get_buf

```

*/

void get_buf(DataVector arg, FuncArg result_cb, SOCKET s);

/*!

* @public Метод реализующий обработку команды отправки управляющих
воздействий на GSM-устройства

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку
данных, например)

* @param s - сокет объекта назначения

* @return None

* @note осуществляет отправку управляющих команд, указанными
пользователем, на GSM-устройства

* @brief (!) НЕОБХОДИМЫ аргументы

* @brief обработчик команды: send_command <args>

*/

void send_command(DataVector arg, FuncArg result_cb, SOCKET s);

// cmd: check_device

/*!

* @public Метод реализующий обработку команды просмотра наличия
подключенных GSM-устройств

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку
данных, например)

* @param s - сокет объекта назначения

* @return None

```

* @note осуществляет проверку наличия подключенных GSM-устройств к серверу.

* @note В качестве ответа отправляется модель GSM-модуля

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: check_device

*/

```
void check_device(DataVector arg, FuncArg result_cb, SOCKET s);
```

```
/*!
```

* @public Метод реализующий обработку команды запроса справки команд сервера

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку данных, например)

* @param s - сокет объекта назначения

* @return None

* @note возвращает список доступных команд с их описанием

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: help

*/

```
void help(DataVector arg, FuncArg result_cb, SOCKET s);
```

```
/*!
```

* @public Тестовый метод

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку данных, например)

```

* @param s - сокет объекта назначения

* @return None

* @note отправляет всем клиентам сервера тестовое сообщение

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: test

*/

void test(DataVector arg, FuncArg result_cb, SOCKET s);

// cmd: store <args>

/*!

* @public Метод реализующий обработку команды хранения данных

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку
данных, например)

* @param s - сокет объекта назначения

* @return None

* @note осуществляет хранения данных на БД сервера

* @brief (!) НЕОБХОДИМЫ аргументы

* @brief обработчик команды: store <args>

*/

void store(DataVector arg, FuncArg result_cb, SOCKET s);

/*!

* @public Метод реализующий обработку команды запроса всех хранящихся
данных

* @param arg - вектор аргументов команды

```


* @param result_cb - функция осуществляющая пост-обработку (отправку данных, например)

* @param s - сокет объекта назначения

* @return None

* @note возвращает список всех данных в БД сервера (помеченные как удаленные и прочитанные, так и нет)

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: get_all

*/

void get_all(DataVector arg, FuncArg result_cb, SOCKET s);

/*!

* @public Метод реализующий обработку команды запроса всех прочитанных хранящихся данных

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку данных, например)

* @param s - сокет объекта назначения

* @return None

* @note возвращает список всех прочитанных данных в БД сервера

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: get_read

*/

void get_read(DataVector arg, FuncArg result_cb, SOCKET s);

/*!

```

* @public Метод реализующий обработку команды запроса всех
непрочитанных хранящихся данных

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку
данных, например)

* @param s - сокет объекта назначения

* @return None

* @note возвращает список всех непрочитанных данных в БД сервера

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: get_unread

*/

void get_unread(DataVector arg, FuncArg result_cb, SOCKET s);

/*!

* @public Метод реализующий обработку команды установки всех
хранящихся данных как прочитанные

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку
данных, например)

* @param s - сокет объекта назначения

* @return None

* @note помечает все хранящиеся данные USB-трафика в БД сервера как
прочитанные

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: set_read_all

*/

```

```

void set_read_all(DataVector arg, FuncArg result_cb, SOCKET s);

/*!

* @public Метод реализующий обработку команды установки всех
хранящихся данных как удаленные

* @param arg - вектор аргументов команды

* @param result_cb - функция осуществляющая пост-обработку (отправку
данных, например)

* @param s - сокет объекта назначения

* @return None

* @note помечает все хранящиеся данные USB-трафика в БД сервера как
удаленные

* @brief (!) НЕ принимает аргументов

* @brief обработчик команды: clear_db

*/

void clear_db(DataVector arg, FuncArg result_cb, SOCKET s);

private:

//
=====

=====

// SERVICE FUNCTIONS

// function: get database connection

/*!

* @public @static Метод возвращающий объект подключения к БД

* @return CDBConnection - объект подключения к БД сервера

*/

```

```

static std::unique_ptr<CDBConnection> get_db();

};

#endif //SERVER_FUNCTIONS_H

/*
=====
=====

* File: scheduler.h

* Description: Планировщик выполнения команд сервера

* Created: 17.02.2017

* Author: soltanoff

*
=====
===== */

#ifndef SCHEDULER_H

#define SCHEDULER_H

/*
=====
===== */

#include "../config/config.h"

#include <regex>

#include <functional>

#include <vector>

#include <map>

#include <list>

```

```

/*
=====

===== */

//! @typedef тип команды

enum class COMMANDS : std::uint32_t {

    unknown=0,

    help,

    test,

    device,

    command,

    clear_db,

    get_all,

    get_buf,

    get_read,

    get_unread,

    rr,

    set_read_all,

    store,

};

/*
=====

===== */

/*!

* @class Scheduler

* Класс осуществляющий планирование выполнения команд сервера

```

```

*/

class Scheduler {

public:

    //! @typedef тип функции пост-обработки
    typedef std::function<void(std::vector<std::uint8_t>, SOCKET)> ResultCallback;

    //! @typedef тип функции обработки команд
    typedef std::function<void(std::vector<std::uint8_t>, ResultCallback, SOCKET)>
Func;

    /*!

    * @public @static Основной метод обработки команд
    * @param cmd_code - код типа команды
    * @param args - аргументы команды
    * @param s - сокет объекта назначения
    * @return true - команда обработана успешно, иначе false
    * @note осуществляет поиск обработчика команды по коду и осуществляет
дальнейшее ее выполнение

    */

    static bool execute_command(const std::uint32_t& cmd_code, const std::string&
args, SOCKET& s);

    /*!

    * @public @static Метод планирования обработки команд
    * @param data - данные полученные от клиента
    * @param args - аргументы команды
    * @param is_gsm - флаг GSM-устройства

```

```

* @return true - команда обработана успешно, иначе false

* @note осуществляет анализ полученных от клиента данных, а также их
дальнейшая обработка

*/

static int schedule(const std::vector<uint8_t> &data, SOCKET s, const bool&
is_gsm);

/*!

* @private Метод связывающий код команды, функцию обработки команд и
результатирующей функции

* @param cmd - код команды

* @param callback - функция обработки команды

* @param result_cb - результирующая функция (функция осуществляющая
пост-обработку (отправку данных, например))

* @note осуществляет связку функции обработки команд с конкретным
объектом-обработчиком

*/

static void bind(COMMANDS cmd, Func&& callback, ResultCallback&&
result_cb);

private:

    //! @private @static словарь callback'ов результирующей функции и функции-
обработчика

    static std::map< std::uint32_t, std::list< std::pair<Func, ResultCallback> > >
_callbacks_map;

};

```

```
#endif /* SCHEDULER_H */
```

```
/*
```

```
=====
=====
```

```
* File: database.h
```

```
* Description: Подключение к базе данных MySQL сервера
```

```
* Created: 17.02.2017
```

```
* Author: soltanoff
```

```
*
```

```
=====
===== */
```

```
#ifndef DATABASE_H
```

```
#define DATABASE_H
```

```
/*
```

```
=====
===== */
```

```
#include <mysql/mysql.h>
```

```
#include <stdio.h>
```

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

```
/*
```

```
=====
===== */
```

```
//! @typedef вектор результатов выполнения запроса
```



```

typedef std::vector<std::string> QueryResult;

//! @typedef список (вектор) векторов результатов выполнения запроса

typedef std::vector<std::vector<std::string>> QueryResultList;

/*
=====
===== */

/*!

* @class CDBConnection

* Класс осуществляющий подключение к БД MySQL

*/

class CDBConnection {

public:

    CDBConnection() {};

    CDBConnection(

        const std::string& server,

        const std::string& database,

        const std::string& user,

        const std::string& password,

        const unsigned int port = 0

    );

    ~CDBConnection() { close(); };

    /*!

    * @public Метод осуществляющий подключение к БД MySQL

    * @param server - ip-адрес сервера БД

```

- * @param database - имя БД
- * @param user - логин пользователя БД
- * @param password - пароль пользователя БД
- * @param port - порт БД
- * @return true если подключение успешное, иначе false
- * @note содержит логику подключения в серверу БД
- */

```
bool connect(
    const std::string& server,
    const std::string& database,
    const std::string& user,
    const std::string& password,
    const unsigned int port = 0
);
```

```
/*!
```

- * @public Метод осуществляющий переподключение к БД MySQL
- * @return None
- * @note содержит логику переподключения в серверу БД
- */

```
void reconnect();
```

```
/*!
```

- * @public Метод осуществляющий закрытие подключения к БД MySQL
- * @return None
- * @note содержит логику закрытия подключения в серверу БД

```

*/

void close();

//! @public объект запроса к БД
MYSQL_RES* query(const char* stmt);

//! @public вектор результатов выполнения запроса
QueryResult get_record(const char* stmt);

//! @public список (вектор) векторов результатов выполнения запроса
QueryResultList get_record_list(const char* stmt);

private:

//! @private основной объект подключения к БД
MYSQL* _connection;

//! @private порт БД
unsigned int _port;

//! @private ip-адрес сервера БД
std::string _server;

//! @private имя БД
std::string _database;

//! @private логин пользователя БД
std::string _user;

//! @private пароль пользователя БД
std::string _password;

};

#endif /* DATABASE_H */

```

```

/*
=====

=====

* File: clientinteraction.h

* Description: Реализация полного взаимодействия с сетевыми клиентами

* Created: 01.02.2017

* Author: soltanoff

*
=====

===== */

#ifndef SERVERTHREAD_H

#define SERVERTHREAD_H

/*
=====

===== */

#include "../config/config.h"

#include "../core/commands/scheduler.h"

#include "../core/commands/functions.h"

#include "interaction.h"

/*
=====

===== */

/*!

* @class ClientInteraction

* @inherit IBaseInteraction

```

* Абстрактный класс описывающий основные поля и методы для реализации
сетевого взаимодействия объекта

*/

class ClientInteraction: public IBaseInteraction {

public:

ClientInteraction(std::uint32_t client_number, SOCKET client_socket, bool
is_gsm); // , std::mutex& server_mutex);

~ClientInteraction() { close(); }

/*!

* @public Основной циклический метод класса.

* @return 0 если работа завершилась без аварий, иначе -1

* @note основной метод сетевого взаимодействия

*/

int exec();

/*!

* @public Метод закрывающий сокет объекта.

* @return None

* @note закрытие сокета осуществляется с параметром SHUT_WR

*/

void close();

private:

//! @private порядковый номер клиента

std::uint32_t _client_number;

//! @private активный сокет клиента

```

SOCKET _client_socket;

///@private объект содержащий функции-обработчики команд сервера

Functions _functions;

///флаг GSM-устройства

bool _is_gsm;

/*!

* @private Метод обработки ответа клиента

* @param bytesRecv - количество полученных байт (out)

* @param answer - массив данных, содержащий ответ (out)

* @return true - данные получены успешно, иначе false

* @note осуществляет обработку полученного ответа клиента по сети

*/

bool get_answer(std::int32_t& bytesRecv, char* answer);

/*!

* @private Метод отправки данных клиенту

* @param msg - массив отправляемых данных

* @return true - данные отправлены успешно, иначе false

* @note осуществляет отправку обработанных данных объекта клиенту по
протоколу TCP/IP

*/

bool send_message(const char* msg);

/*!

* @private Метод отправки данных объекту по сокету

* @param msg - массив отправляемых данных

```

```

* @param s -

* @return true - данные отправлены успешно, иначе false

* @note осуществляет отправку обработанных данных объекта клиенту по
протоколу TCP/IP

*/

bool send_message(const char* msg, SOCKET s);

/*!

* @private Метод связывающий функции обработки команд с объектом-
обработчиком

* @param fptr - функция обработки команды

* @param optr - объект-обработчик

* @note осуществляет связку функции обработки команд с конкретным
объектом-обработчиком

*/

template<class TFunc, class TObj>

auto bind(TFunc&& fptr, TObj& optr) -> decltype(std::bind(fptr, optr,
std::placeholders::_1, std::placeholders::_2, std::placeholders::_3)) {

    return std::bind(fptr, optr, std::placeholders::_1, std::placeholders::_2,
std::placeholders::_3);

}

};

/*

=====

===== */

#endif /* SERVERTHREAD_H */

```

```

/*
=====
=====

* File: connection.h

* Description: Базовый интерфейс описывающий подключение к сети

* Created: 01.02.2017

* Author: soltanoff

*
=====
===== */

#ifndef CONNECTION_H
#define CONNECTION_H

/*
=====
===== */

#include "../config/config.h"

/*
=====
===== */

/*!

* @interface IBaseConnection

* Абстрактный класс описывающий основные поля и методы для реализации
подключения по протоколу TCP/IP

*/

class IBaseConnection {

public:

```



```

virtual ~IBaseConnection() = default;

/*!

* @public Метод осуществляющий инициализацию сокета и дальнейшего
подключения к серверу.

* @return 0 если открыть сокет получилось, иначе -1

* @note Содержит в себе логику настройки сетевого подключения

*/

virtual int try_open_socket() = 0;

/*!

* @public Метод позволяющий начать работу сетевой части.

* @return None

* @note запуск метода IBaseConnection::try_open_socket()

*/

virtual void start() = 0;

protected:

    //! @protected сокет открытого подключения по протоколу TCP/IP
    SOCKET m_socket;

    //! @protected структура конфигураций подключения сети
    sockaddr_in service;

};

/*
=====
===== */

#endif /* CONNECTION_H */

```

```

/*
=====

=====

* File: interaction.h

* Description: Базовый интерфейс описывающий сетевое взаимодействие

* Created: 01.02.2017

* Author: soltanoff

*
=====

===== */

#ifndef INTERACTION_H

#define INTERACTION_H

/*
=====

===== */

/*!

* @interface IBaseInteraction

* Абстрактный класс описывающий основные поля и методы для реализации
сетевого взаимодействия объекта

*/

class IBaseInteraction{

public:

    virtual ~IBaseInteraction() = default;

    /*!

    * @public Основной циклический метод класса.

```

```

* @return 0 если работа завершилась без аварий, иначе -1
* @note основной метод сетевого взаимодействия
*/
virtual int exec() = 0;

/*!
* @public Метод закрывающий сокет объекта.
* @return None
* @note закрытие сокета осуществляется с параметром SHUT_WR
*/
virtual void close() = 0;

protected:

/*!
* @protected Метод обработки ответа
* @param bytesRecv - количество полученных байт (out)
* @param answer - массив данных, содержащий ответ (out)
* @return true - данные получены успешно, иначе false
* @note осуществляет обработку полученного ответа по сети
*/
virtual bool get_answer(int& bytesRecv, char* answer) = 0;

/*!
* @protected Метод отправки данных
* @param msg - массив отправляемых данных
* @return true - данные отправлены успешно, иначе false

```

* @note осуществляет отправку обработанных данных объекта в пункт назначения по протоколу TCP/IP

```
*/

virtual bool send_message(const char* msg) = 0;

};

/*
=====
===== */

#endif /* INTERACTION_H */

/*
=====
=====

* File: server.h

* Description: Реализация работы сервера

* Created: 01.02.2017

* Author: soltanoff

*
=====
===== */

#ifndef SERVER_H

#define SERVER_H

/*
=====
===== */

#include "../core/clientinteraction.h"

#include "../core/connection.h"
```

```

/*
=====
===== */

//! вектор сокетов подключенных к серверу GSM-устройств
extern std::vector<SOCKET> GSM_MODULES_SOCKETS;

//! @typedef тип подключенного клиента
enum class CLIENT_TYPES : std::uint32_t {

    simple=1,

    gsm

};

/*!

* @class CServer

* @inherit IBaseConnection

* Класс осуществляющий работу сервера

*/

class CServer: public IBaseConnection {

public:

    CServer();

    ~CServer();

    /*!

    * @public Метод осуществляющий инициализацию сокета и дальнейшего
    подключения к серверу.

    * @return 0 если открыть сокет получилось, иначе -1

    * @note Содержит в себе логику настройки сетевого подключения

```

```

*/

int try_open_socket();

/*!

* @public Основной циклический метод класса.

* @return 0 если работа завершилась без аварий, иначе -1

* @note основной метод для взаимодействия сервера с клиентами

*/

int exec();

/*!

* @public Метод позволяющий начать работу серверной части.

* @return None

* @note запуск методов CClient::try_open_socket(), CClient::exec()

*/

void start();

/*!

* @public Метод закрывающий сокет сервера.

* @return None

* @note закрытие сокета с параметром SHUT_WR

*/

void close();

private:

    //! @private вектор умных указателей на handler'ы потоков клиентов
    std::vector<std::shared_ptr<std::thread>> _client_threads;

    //! @private вектор активных сокетов клиентов

```

```

std::vector<SOCKET> _client_sockets;

///  

std::vector<char*> _client_ips;

/*!  

 * @private Метод определяющий является ли подключенный клиент  

устройством GSM.  

 * @param AcceptSocket - сокет подключенного клиента  

 * @param client_number - номер подключенного клиента  

 * @return true - подключилось устройство GSM, иначе false  

 * @note осуществляет подтверждение подключения клиента к серверу  

(отправляет клиенту статус подключения)  

 * @note определяет тип подключенного клиента  

 */  

bool is_gsm_client(const SOCKET& AcceptSocket, const int& client_number);

/*!  

 * @private Метод осуществляющий подключение клиента к серверу на  

уровне сокетов.  

 * @param AcceptSocket - сокет подключенного клиента  

 * @param ClientInfo - структура содержащая основную информацию о  

подключенном клиенте  

 * @return None  

 * @note осуществляет подтверждение подключения клиента к серверу на  

уровне сокетов  

 */  

void accept_socket(SOCKET& AcceptSocket, sockaddr_in& ClientInfo);

```

```

/*!

* @private Метод регистрирующий подключение клиента .

* @param AcceptSocket - сокет подключенного клиента

* @param ClientInfo - структура содержащая основную информацию о
подключенном клиенте

* @param count - значение счетчика клиентов

* @return None

* @note осуществляется регистрация: сокета, ip-адреса клиента, handler'a
потока обработки данных клиента

*/

void connect_user(const SOCKET& AcceptSocket, const sockaddr_in&
ClientInfo, const std::uint32_t& count);

};

/*
=====
===== */

#endif /* SERVER_H */

/*
=====
=====

* File: main.cpp

* Description: Основная функция проекта

* Created: 01.02.2017

* Author: soltanoff

```



```

*
=====
===== */

#ifdef _WIN32

#include "src/client/client.h"

#else

#include "src/server/server.h"

#include "src/client/client.h"

#endif

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

    time_t t = time(NULL);

    tm *aTm = localtime(&t);

    std::cout

        << "[START] TIME: "

        << aTm->tm_year + 1900 << "/" << std::setfill('0') << std::setw(2)

        << aTm->tm_mon + 1 << "/" << std::setfill('0') << std::setw(2)

        << aTm->tm_mday << " " << std::setfill('0') << std::setw(2)

        << aTm->tm_hour << ":" << std::setfill('0') << std::setw(2)

        << aTm->tm_min << ":" << std::setfill('0') << std::setw(2)

        << aTm->tm_sec << std::endl;

#ifdef _WIN32

    std::cout << "This server build not for windows!" << std::endl;

```

```
    system("pause");  
#else  
    CServer s;  
    CClient s;  
    s.start();  
#endif  
    return 0;  
}
```

ПРИЛОЖЕНИЕ Г

```
CREATE TABLE `Logger` (  
    `id` int(11) NOT NULL AUTO_INCREMENT,  
    `date` datetime DEFAULT CURRENT_TIMESTAMP COMMENT 'Дата создания  
записи',  
    `message` varchar(4048) DEFAULT NULL COMMENT 'USB-трафик',  
    PRIMARY KEY (`id`)  
)  
  
ENGINE = INNODB  
  
COMMENT = 'Таблица для хранения USB-трафика';
```

```
CREATE TABLE `State` (  
    `id` int(11) NOT NULL AUTO_INCREMENT,  
    `id_logger` int(11) DEFAULT NULL COMMENT 'id USB-трафика {Logger}',  
    `modify_date` datetime DEFAULT NULL COMMENT 'Дата изменения записи',  
    `is_read` tinyint(1) DEFAULT 0 COMMENT 'Флаг непрочитанной записи',  
    `deleted` tinyint(1) DEFAULT 0 COMMENT 'Флаг удаленной записи',  
    PRIMARY KEY (`id`),  
    CONSTRAINT `FK_state_id_logger` FOREIGN KEY (`id_logger`)  
    REFERENCES `Logger` (`id`) ON DELETE NO ACTION ON UPDATE  
    RESTRICT  
)
```

ENGINE = INNODB

COMMENT = 'Таблица состояния USB-трафика';

DELIMITER \$\$

CREATE TRIGGER InsertStateForLogRow AFTER INSERT ON Logger

FOR EACH ROW

BEGIN

INSERT INTO State (id_logger) VALUES

(new.id);

END; \$\$

DELIMITER ;

ПРИЛОЖЕНИЕ Д

```
cmake_minimum_required(VERSION 2.0)
```

```
set(CMAKE_CXX_STANDARD 14)
```

```
if(WIN32)
```

```
    project(client)
```

```
    set(SOURCE_FILES
```

```
        main.cpp
```

```
        src/client/client.cpp
```

```
        src/config/config.cpp)
```

```
    add_executable(client ${SOURCE_FILES})
```

```
    target_link_libraries(client wsock32 ws2_32)
```

```
else()
```

```
    project(server)
```

```
    set(SOURCE_FILES
```

```
        main.cpp
```

```
        src/server/server.cpp
```

```
        src/client/client.cpp
```

```
        src/core/clientinteraction.cpp
```

```
        src/config/config.cpp
```

```
        src/core/commands/functions.cpp
```

```
        src/core/commands/scheduler.cpp
```

```
src/core/database/database.cpp
```

```
src/core/commands/functions.h)
```

```
add_definitions(-std=c++14 -O0 -g)
```

```
add_executable(server ${SOURCE_FILES})
```

```
target_link_libraries(server pthread mysqlclient)
```

```
endif()
```

ПРИЛОЖЕНИЕ E

```
cmake_minimum_required(VERSION 2.0)

set(CMAKE_CXX_STANDARD 14)

if(WIN32)

    project(client)

    set(SOURCE_FILES

        main.cpp

        src/client/client.cpp

        src/config/config.cpp)

    add_executable(client ${SOURCE_FILES})

    target_link_libraries(client wsock32 ws2_32)

else()

    project(client)

    set(SOURCE_FILES

        main.cpp

        main.cpp

        src/client/client.cpp

        src/config/config.cpp)

    add_definitions(-std=c++14 -O0 -g)

    add_executable(client ${SOURCE_FILES})

endif()
```

ПРИЛОЖЕНИЕ Ж

```
#!/bin/bash
```

```
cd /server
```

```
cmake clean
```

```
make clean
```

```
cmake .
```

```
make
```

```
systemctl daemon-reload
```


ПРИЛОЖЕНИЕ 3

```
#!/bin/bash
```

```
cd /client
```

```
cmake clean
```

```
make clean
```

```
cmake .
```

```
make
```

```
systemctl daemon-reload
```