МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___» _____ 2022 г.

ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС СБОРА ДАННЫХ ТЕХНИЧЕСКОГО СОСТОЯНИЯ ТРАНСПОРТНОГО СРЕДСТВА

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2022.247 ПЗ ВКР

Руководитель работы,
к.п.н., доцент каф. ЭВМ
_____Ю.Г. Плаксина
«___»_____2022 г.

Автор работы,
студент группы КЭ-406
_____В.А. Попов
«___»_____2022 г.

Нормоконтролёр,
к.п.н., доцент каф. ЭВМ
_____М.А. Алтухова
«___»_____2022 г.

Челябинск-2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2022 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Попову Всеволоду Андреевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Программно-аппаратный комплекс сбора данных технического состояния транспортного средства» утверждена приказом по университету от __ декабря 2021 г. №____

2. **Срок сдачи студентом законченной работы:** «1» июня 2022 г.

3. **Исходные данные к работе:**

– операционная система Linux, Windows 10;

– язык программирования C++, Python3;

– среда разработки Visual Studio, Thonny IDE;

– микрокомпьютер Raspberry Pi3 model B+;

– диагностический адаптер ELM32-PL2303 с USB интерфейсом.

4. **Перечень подлежащих разработке вопросов:**

– рассмотрение существующих комплексов для диспетчеризации и сбора данных технического состояния транспортного средства;

– анализ современных аппаратных технологий диспетчеризации и сбора данных технического состояния транспортного средства;

– проектирование и реализация собственного программно-аппаратного комплекса для диспетчеризации и сбора данных технического состояния транспортного средства;

– оценка работоспособности разработанного программно-аппаратного комплекса в различных режимах и внешних условиях.

5. **Дата выдачи задания:** 1 декабря 2021 г.

Руководитель работы _____/ Ю.Г. Плаксина /

Студент _____/ В.А. Попов /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	07.02.2022	
Разработка модели, проектирование	07.03.2022	
Реализация системы	04.04.2022	
Тестирование, отладка, эксперименты	10.05.2022	
Компоновка текста работы и сдача на нормоконтроль	16.05.2022	
Подготовка презентации и доклада	24.05.2022	

Руководитель работы _____/ Ю.Г. Плаксина /

Студент _____/ В.А. Попов /

АННОТАЦИЯ

В.А. Попов. «Программно-аппаратный комплекс сбора данных технического состояния транспортного средства». – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2022, 55 с., 15 ил., библиогр. список – 19 наим.

В рамках выпускной квалификационной работы производится обзор, анализ, выявление плюсов и минусов существующий программно-аппаратных решений для сбора и обработки данных технического состояния автотранспорта. Осуществляется разработка программно-аппаратного комплекса сбора и анализа данных технического состояния автомобиля для внедрения его в уже готовую систему предприятия. Подробно рассматриваются способы коммуникации между диагностическим контроллером и электронным блоком управления автомобиля, между диагностическим контроллером и компьютером. Доказывается актуальность проблемы и нехватка программно-аппаратных решений на внутреннем рынке.

Цель разрабатываемой аппаратной части заключается в создании бюджетного и универсального устройства для сбора данных технического состояния автомобиля и отправки их на устройство-сервер, интегрируемый в систему предприятия.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ГЛОНАСС – глобальная навигационная спутниковая система

ГП – графический процессор

ДТП – дорожно-транспортное происшествие

ОЗУ – оперативное запоминающее устройство

ОС – операционная система

ПО – программное обеспечение

ТС – транспортное средство

ЦП – центральный процессор

ALDL (англ. Assembly Line Diagnostic Link) – проприетарная бортовая диагностическая система автоконцерна «General Motors»

API (англ. Application Programming Interface) – программный интерфейс приложения

DTC (англ. Diagnostic Trouble Code) – диагностический код неисправности

GPS (англ. Global Positioning System) – спутниковая система навигации

KBps (англ. Kilobit per second) – килобит в секунду

OBD (англ. On-Board Diagnostic) – бортовая диагностика

ВВЕДЕНИЕ

Современный автомобиль является сложным высокотехнологическим устройством. Внутри него между собой взаимодействуют высокоточные механические и электронные системы, с помощью которых и происходит корректная работа автотранспорта. При возникновении неисправностей следует провести оценку технического состояния автомобиля и только после этого необходимо устранить неисправность.

Оценка технического состояния и определение неисправностей автотранспортного средства называется диагностика. От её качества напрямую зависит фронт работ и финансовые затраты на устранение проблемы.

Диагностика бывает двух видов: техническая и электронная. Техническая диагностика подразумевает наличие специалиста, обладающего определенными знаниями и навыками в обслуживании транспортного средства. Электронная диагностика осуществляется при помощи подключения специального устройства к электронному блоку управления двигателем для считывания информации об ошибках и данных с модулей и датчиков автомобиля.

В нашей стране остро стоит вопрос обслуживания автомобилей и его качество. Учитывая финансовое положение, любая поломка может оказаться весьма затратной. Также необходимо принять во внимание то, что неполадки в системах транспортного средства могут привести к дорожно-транспортным происшествиям.

Доля ДТП, произошедшие из-за неисправностей отдельных узлов автомобиля, составляет около 14% от общего числа. На рисунке 1 представлены данные 14%: 50% из-за неисправностей тормозного управления, 27% из-за проблем с рулевым управлением, 11% связаны с неисправностями в оптике, 8% составляют неисправности в габаритах или указателях поворота и 4% составляют проблемы с шинами [1].

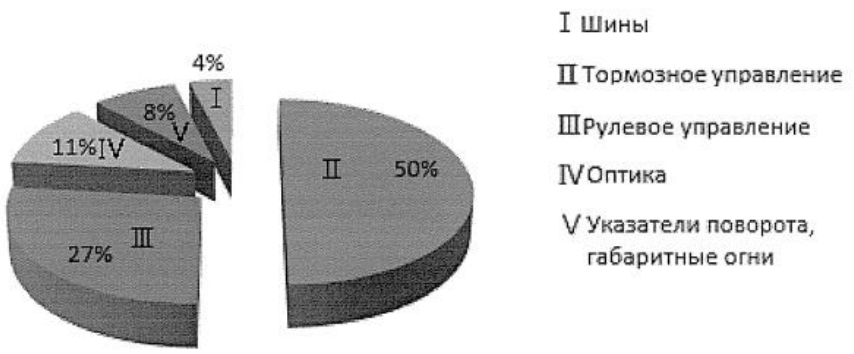

Рисунок 1 – Причины ДТП по техническим неисправностям

В основном массовыми проблемами технического состояния автомобиля обладают таксопарки, что является абсолютной халатность, так как водители такси в первую очередь отвечают за сохранность пассажира, а технические неполадки могут привести к ДТП со смертельным исходом.

Очень сильно этому подвержен коммерческий транспорт по причине того, что водитель никак не участвует в техническом обслуживании и диагностике автомобиля, что сказывается на качестве эксплуатации ТС. Из-за небрежного отношения к автомобилю в первую очередь страдает бюджет автопарка и компании в целом.

Уменьшив время между поломкой и её обнаружением, можно не только спасти жизни водителей, пассажиров и пешеходов, но также уменьшить средства, затраченные на ремонт.

Целью данной выпускной квалификационной работы является разработка программно-аппаратного комплекса сбора данных технического состояния транспортного средства, с помощью которых можно своевременно выявить серьезные технические неисправности автомобиля и предотвратить потенциальные поломки. Данная система предназначена для внедрения в уже готовую систему предприятия и может использоваться как частными лицами, так и фирмами. Планируется, что главная особенность данного программно-аппаратного комплекса является её универсальность и экономичность.

Для достижения поставленной цели необходимо решить следующие задачи:

– рассмотреть существующие на рынке аналоги;

– провести детальный анализ найденных решений, их особенности, минусы и плюсы;

– определить основной функционал программно-аппаратного комплекса;

– выполнить программную реализацию комплекса;

– определить методы тестирования разработанного программно-аппаратного комплекса.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Электронная диагностика автомобилей берет свое начало с 1980-ых годов, когда «General Motors» представила ALDL – система бортовой диагностики автомобиля [2]. Данная система использовалась производителями ТС для диагностики в своих дилерских центрах, но она была не стандартизированной и имела кучу недочетов.

Позже в 1984 году в Калифорнии вышел закон об OBD-I, который распространялся на все новые автомобили, выпускаемые с 1988 в Калифорнии. В этом же году разрабатывается проект OBD-II и уже в 1994 ужесточаются требования к производителям и требует от них установку OBD-II интерфейса. К 1996 году данный закон работает в полную силу и действует на все автомобили, проданные на территории США, и уже сегодня данный интерфейс является обязательным для производителей всех стран [3]. Сам стандарт OBD-II предоставляет полный контроль над двигателем, позволяет проводить мониторинг датчиков и сети управления ТС, что упрощает работу по его обслуживанию и настройки.

В наше время получать данные из электронного блока управления при помощи OBD-II интерфейса могут не только в специализированных сервисах, но и обычные люди при помощи специальных адаптеров, которые можно легко купить в автомагазине или же заказать в интернете за небольшую цену.

В данной выпускной квалификационной работе планируется разработать систему сбора данных технического состояния автомобиля при помощи интерфейса OBD-II. Считывать необходимую информацию планируется при помощи диагностического сканера ELM327 с USB интерфейсом. Данный сканер подключается к микрокомпьютеру, который в свою очередь собирает и отправляет данных о техническом состоянии автомобиля на сервер. Основной же проблемой является передача данных от транспортного средства до сервера, но учитывая то, как развились информационно-коммуникационные сети, которые позволяют передавать

данные в любую точку мира, имеется возможность организовать передачу необходимой информации с помощью сети-Интернет с использованием TCP/IP протоколов.

Системы диспетчеризации технических данных автомобиля и системы телематики нашли свое применение в грузоперевозках, таксопарках и каршерингах [4, 5].

1.1 Обзор аналогов

В наше время на Российском рынке представлено не так много программно-аппаратных комплексов. Ниже представлены компании, плюсы и минусы систем и их описание.

Компания «Спрут-технология» занимается разработкой и внедрением систем в область автоматизации подготовки производства [6]. Одним из главных продуктов компании является система «Geostron». Это программно-аппаратный комплекс для дистанционного мониторинга технического состояния автомобиля и его геолокации с помощью GPS и ГЛОНАСС. Данная система применяется в области транспортной логистики, сельского хозяйства, служб такси и каршеринга, городского транспорта и во многих других отраслях [6].

Интерфейс веб-версии приложения системы представлен на рисунке 2. На рисунках 3 и 4 представлены устройства Teltonika FMB 010 и FMB 003 [7, 8], которые являются прямыми аналогами разрабатываемой системы. Они также подключаются к OBD-II интерфейсу и также передают данные при помощи сети интернет через мобильную сеть. Их главным преимуществом является компактность, они не требуют какого-либо дополнительного закрепления к автомобилю, помимо обычного подключения к интерфейсу.

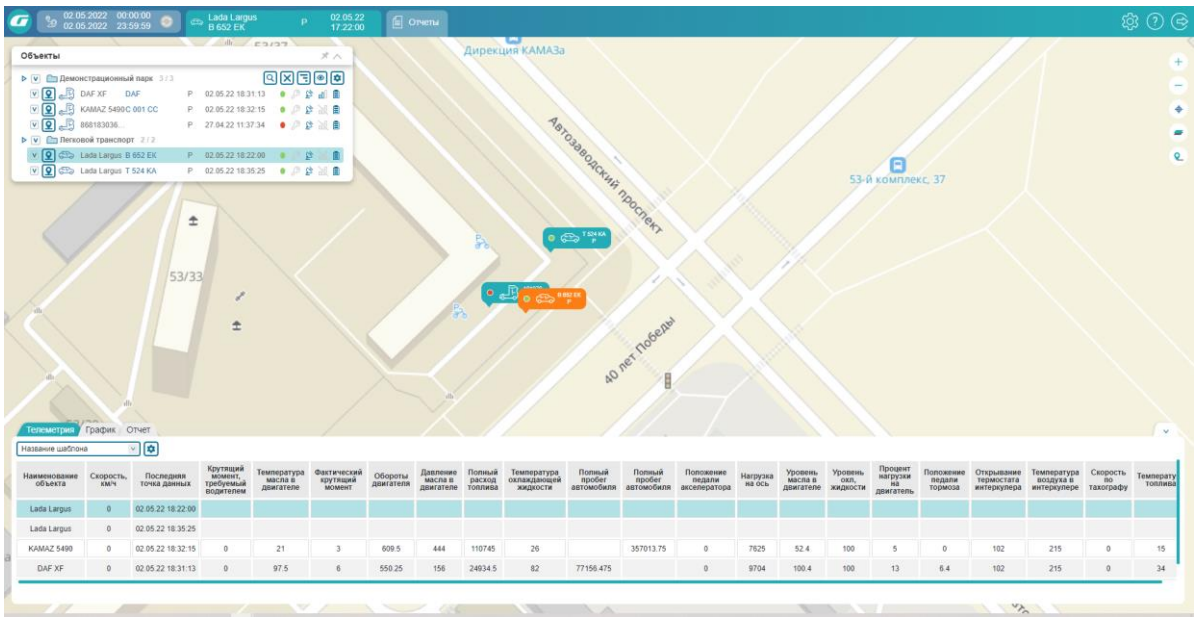

Рисунок 2 – Интерфейс веб-версии geostron

Рисунок 3 – Teltonika FMB 010

Рисунок 4 – Teltonika FMB 003

Плюсы:

– мобильность и компактность устройств;

– высокая точность;

– определение местоположения с помощью систем GPS/ГЛОНАСС;

– удобный интерфейс.

Минусы:

– компания предоставляет свою готовую систему и не производит внедрение в системы предприятия, занимающиеся транспортными перевозками, каршерингами и таксопарками;

– высокая стоимость;

– нет возможности добавить большего функционала к терминалу клиентом.

Система «TrackGPS» специализируется в первую очередь на получение геоданных, контроль маршрутов транспорта, информации о расходе топлива и другие технические данные состояния ТС [9]. Данная система используется в строительных организациях и многих других областях.

В каталоге компании имеются трекеры для автомобиля, подключаемые напрямую к CAN-шине. Также имеются трекеры с OBD-II интерфейсом, но они сняты с производства.

Плюсы:

– мобильность и компактность устройств;

– определение местоположения с помощью систем GPS/ГЛОНАСС.

Минусы:

– нет трекера с интерфейсом OBD-II;

– нет возможности добавить большего функционала к терминалу клиентом.

Компания «Монтранс» является разработчиком программно-аппаратных систем для автоматизации транспортных компаний, телематических систем и оборудования [10]. Компания разрабатывает

системы для областей спецтехники, грузового и легкого коммерческого транспорта.

Компания предоставляет сбор технических данных, например, контроль горючего, просмотр моточасов, контроль пробега и геозон. Как и другие компании, «Монтранс» использует системы GPS/ГЛОНАСС для мониторинга геопозиции автотранспорта.

К сожалению, компания не предоставляет технических характеристик своего оборудования и крайне сложно определить плюсы и минусы данной системы.

«Cartrek» – программно-аппаратный комплекс для сбора технических и гео-данных [11]. Компания специализируется в областях каршеринга, таксопарках и аренды самокатов.

К сожалению, как и компания «Монтранс», «Cartrek» не предоставляет технических характеристик своего оборудования. Единственный плюс, который можно выявить из доступной информации – это внедрение в готовые системы предприятий при помощи API.

1.2 Анализ основных технологических решений

1.2.1 Выбор языков программирования

Ниже представлены рассматриваемые языки программирования для разработки ПО терминала и сервера.

C++ является производительным языком программирования. Производительность – самая главная характеристика для любого сервера, так как ежеминутно через него проходят тысячи операций. Основным средством данного языка, с помощью которого он справляется со множеством математических операций в секунду – это многопоточность. Также основным плюсом C++ является работа на низком уровне взаимодействия с ядром. Это позволяет программе работать напрямую с памятью, адресами и портами.

Помимо этого, при помощи шаблонов C++ позволяет создавать обобщенные алгоритмы для разных типов данных и их вычисление на уровне компиляции [12].

Несмотря на это, C++ имеет и свои минусы. Основной минус языка – высокий порог вхождения. Из-за того, что данный язык является низкоуровневым, он требует от программиста более четкую работу с памятью. От этого возникают сложно уловимые ошибки, о который программист, в силу своей неопытности, может не заметить. Также реализация объектно-ориентированной программы может замедлить быстродействие кода. Стоит признать, что C++ унаследовал много проблем от своего предшественника языка C.

C#. Данный язык программирования принято считать наследником C++. На данный момент он является довольно популярным языком и быстроразвивающимся и от этого и вытекают все плюсы и минусы [13].

Основным плюсом C# является огромное количество утилит с использованием только дотнета. Язык обладает очень легким синтаксисом и огромной поддержкой компании разработчика и сообщества. Также C# обладает очень хорошей поддержкой объектно-ориентированного программирования, ибо сама структура кода полностью зависит от классов и методов. Порог вхождения у данного языка довольно низкий, а его синтаксис очень похож на синтаксисы других интерпретируемых языков программирования, например, на Java.

Из минусов хотелось бы подчеркнуть то, что C# не обладает быстрой скоростью, как C++, но тем не менее является самым быстрым в своей категории благодаря преобразования кода в промежуточный язык во время компиляции. Также возникают сложности при написании программ под Linux-системы.

Java считается языком общего назначения. Несмотря на свой возраст, он до сих пор считается весьма популярным и обладает хорошей поддержкой. Как и C#, Java использует концепцию объектно-ориентированного

программирования и является высокоуровневым, что делает синтаксис легче, что в свою очередь понижает порог вхождения. Java является безопасным языком. Как и большинство высокоуровневых языков обладает низкой скоростью из-за интерпретации кода.

Python на данный момент является очень популярным решением для начинающих программистов в силу простого синтаксиса, обладает обширной коллекцией сред разработки. Крайне универсален из-за своей кроссплатформенности и поэтому востребован на рынке. Основным минусом является низкая скорость, как и у других интерпретируемых языков. Также стоит взять во внимание ограниченность работы с базами данных.

Основной плюс данного языка – простота. Одно из самых главных качеств, которые пригодились бы для написания терминала, ведь он принимает и отправляет информацию по виртуальному COM-порту. На других языках программирования это бы вызвало сложности при написании программы, но для питона имеется большое количество библиотек, которые позволяют облегчить написание подобных программ. К примеру, в нашем проекте используется библиотека Pyserial, которая будет обрабатывать и отправлять данные для COM-порта. Он быстр и прост и это его основной плюс.

1.2.2 Выбор устройства терминала

Для разработки терминала больше всего подходят одноплаточные микрокомпьютеры, к примеру, Raspberry Pi 3. Данный микрокомпьютер является самым популярным на рынке и считается первопроходцем в мире дешевых одноплаточных решений. Его главный плюс – это доступная вычислительная мощность при небольшой цене и маленьком размере. Но Raspberry Pi является не одним микрокомпьютером на рынке. Ниже приведем примеры основных одноплаточных микрокомпьютерных решений [14].

Raspberyy Pi 3 Model B+ является самым популярным решением на рынке. Имеет на борту процессор Cortex-A53 с частотой 1,2 ГГц и 1,5 Гб ОЗУ, USB, Ethernet, Bluetooth и WiFi интерфейсы, видео выход HDMI.

Orange Pi является самым удачным конкурентом Raspberry Pi. В отличии от Raspberry Pi 3 Model B+, имеет на борту 2 Гб ОЗУ и видеоускоритель Mali-450 GPU, который позволяет воспроизводить видео с качеством до 2K. Также имеется ИК-приемник и возможность управления с помощью ДУ-пульта. Также, помимо стандартных USB, Ethernet, Bluetooth и WiFi интерфейсов, имеется видеоинтерфейс CSI.

Banana Pi M3 микрокомпьютер построенный на базе восьмиядерного SoC A83T с частотой до 1,8 ГГц, который является ЦП ARM Cortex A7 соединенный с ГП SGX544MP1. Имеет данный компьютер 2 Гб ОЗУ и 8 Гб флэш-памяти. Также присутствуют все те же стандартные интерфейсы, которые есть и на других рассматриваемых микрокомпьютерах. Главное отличие – наличие SATA и I2C интерфейсов.

Для данного проекта может подойти не только любая модель Raspberry Pi или его аналога, но и самый обычный персональный компьютер или ноутбук, поскольку основным требованием к терминалу является возможность поддерживания операционной системы Linux и наличие USB-интерфейсов.

Выбор терминального устройства будет зависеть от требуемой клиентом мощности системы, так как предполагается, что помимо нашей программы, на терминале может работать и стороннее ПО.

1.2.3 Выбор диагностического адаптера ELM-327

В данной выпускной квалификационной работе планируется использование адаптера ELM-327 для общения терминала и электронного блока управления автомобиля. На рынке существует множество вариантов данного диагностического адаптера: USB-версия, WIFI-версия и Bluetooth-

версия. Как можно понять – все они отличаются интерфейсом подключения к устройству-терминалу, при этом тип подключения к блоку управления автомобиля остается неизменным, а именно при помощи интерфейса OBD-II.

Для нашего программно-аппаратного комплекса будет использоваться USB-версия адаптера ELM-327 по причине того, что в системах, где не планируется частое обслуживание аппаратной составляющей, проводной интерфейс является более стабильным вариантом подключения, в отличие Bluetooth и WiFi-вариантов.

После выбора интерфейса подключения контроллера к терминалу возникает новый вопрос – выбор основного чипа адаптера. Ниже были рассмотрены основные вариаций чипов USB-адаптера ELM-327.

PIC18F2480 FT232Rl является самой дорогой и качественной моделью из рассматриваемых. Самым главным преимуществом является высокая скорость соединения – 500 KBps, что в 13 раз больше скорости других чипов. Также данный чип отличается высокой стабильностью и надежностью. Он является универсальным решением, т. к. поддерживает работу с электронными блоками управления автомобилей практически всех марок и программами для диагностики. Единственные минусы – его цена и малый объем производства, что осложняет его приобретение [15].

PIC18F25K80 PL2303, в отличии от чипа PIC18F2480, имеет меньшую цену и меньшую скорость соединения. Также, как и его более дорогая версия, поддерживает работу с электронными блоками управления автомобиля практически всех марок и может работать с большим количеством программного обеспечения для диагностики [15]. Является оптимальным недорогим решением, которое отлично подойдет для разрабатываемого программно-аппаратного комплекса.

CH340 является самым дешевым вариантом из рассматриваемых. Низкая цена обуславливается низким качеством продукта и невозможностью подключения ко множеству электронных блоков управления автомобилей разных марок [16]. Необходимо отметить, что во время тестирования

терминала из строя вышли два из двух ELM-327 адаптеров с рассматриваемым чипом. Основная проблема, с которой пришлось столкнуться при тестировании, заключается в невозможности выбора скорости COM-порта для соединения.

1.3 Вывод

Из обзора малого числа аналогичных решений, представленных на отечественном рынке, можно сделать вывод, что большинство компаний не предоставляют возможность интеграции в уже готовые системы предприятий и работают параллельно с ними. Еще необходимо отметить, что далеко не все компании готовы предоставить полную характеристику своих устройств, что затрудняет их обзор и примерное представление о стоимости того или иного продукта.

Подводя итоги анализа технических решений, можно с уверенностью сказать, что из рассматриваемых вариантов языков программирования, для разработки серверной части больше всего подходит C++ из-за своей скорости и оптимизированной работы многопоточного программирования, так как это поможет ускорить обмен информацией сразу с несколькими удаленными терминалами. Для программирования сервера на C++ будет использоваться Visual Studio в силу его простоты работы с разными библиотеками, которые необходимы в данном проекте.

Для разработки терминала данного программно-аппаратного комплекса больше всего подходит язык Python. Отчасти это связано с тем, что тестовый вариант комплекса разрабатывается на платформе микрокомпьютера Raspberry Pi, который в свою очередь нацелен на работу с данным языком программирования. Также можно добавить, что легкость и возможности данного языка сильно облегчают разработку программной части терминала, отвечающую за его связь с диагностическим адаптером ELM-327.

Основным устройством терминала, как указано выше, может быть любой персональный компьютер, поддерживающий USB-интерфейс и ОС Linux. Выбор Raspberry Pi больше связан с тем, что имеется опыт работы с данной платформой.

Чипом для диагностического адаптера ELM-327 был выбран PIC18F25K80 PL2303 из-за поддержки большого количества автомобилей и его цены.

2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

Ниже приведены минимальные системные требования программно-аппаратного комплекса.

Сервер:

– операционная система: Windows 10 (1809 и выше);

– процессор: Intel Core i5 4470, LGA1150;

– место на диске: от 5Gb.

Терминал:

– операционная система: Linux (Kernel 4 и выше);

– процессор: Intel Atom Z2760;

– место на диске: от 1Gb;

– диагностический адаптер: ELM327-USB PL2303;

– USB-модем или иное средство для доступа в сеть.

Автомобиль:

– наличие и исправность OBD-II разъема;

– год выпуска: от 2004.

2.1 Функциональные требования для сервера

Функциональные требования для сервера:

– получение технических данных с терминала;

– обработка полученных данных;

– составление отчета;

– проверка связи с терминалом;

– отправка данных на локальный сокет для работы со сторонними приложениями и системами предприятия.

2.2 Функциональные требования для терминала

Функциональные требования для терминала:

– чтение кодов неисправности при помощи контроллера ELM327;

– расшифровка полученных технических данных состояния автомобиля;

– отправка полученных технических данных состояния автомобиля на сервер;

– режим логирования;

– проверка связи с диагностическим адаптером;

– проверка связи с сервером.

2.3 Нефункциональные требования

Данный программно-аппаратный комплекс предназначен для внедрения в уже готовую систему предприятия. Исходя из этого, клиент может устанавливать дополнительный софт в терминал. Планируется использовать не все ресурсы микрокомпьютера, чтобы оставить резервную мощность для непредусмотренного ПО.

3 ПРОЕКТИРОВАНИЕ

3.1 Архитектура предлагаемого решения

В основе работы программно-аппаратного комплекса лежит модель взаимодействия клиент-сервер. В данном проекте клиентом служит устройство-терминал, получающее технические данные с электронного блока управления и отправляющее их на сервер. Предполагается, что клиент и сервер работают внутри одной виртуальной частной сети (VPN) и взаимодействуют друг с другом при помощи сети Интернет.

Для установления соединения между терминалами и сервером планируется использование технологии сокет-соединений. Сокет – это программный интерфейс, позволяющий процессам обмениваться информацией [17].

Для языков программирования C++ и Python существуют библиотеки для работы с сокетами. Для C++ планируется использовать библиотеку Winsock [18], а для языка Python будем использовать библиотеку Socket. Данные библиотеки позволяют получать доступ и программировать сетевые сервисы, такие как TCP/IP и UDP/IP. С помощью данных библиотек, планируется устанавливать TCP/IP соединение между сервером и терминалами.

На рисунке 5 представлена схема сетевого подключения сервера и терминала.

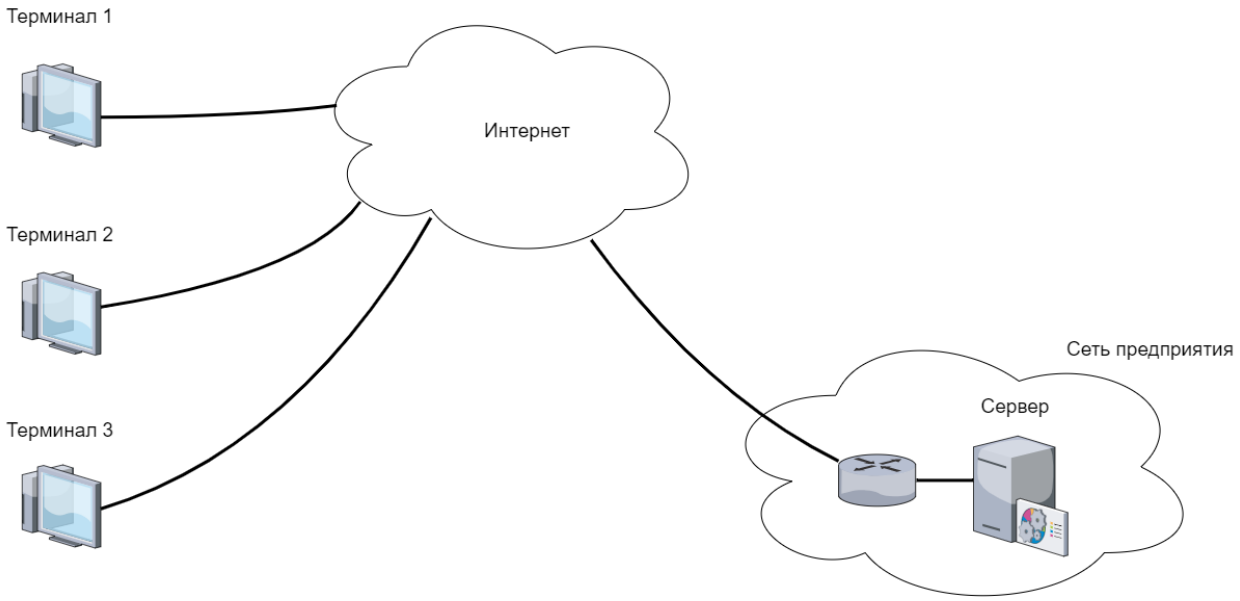

Рисунок 5 – Схема сетевого соединения терминалов и сервера

3.2 Алгоритмы решения задач

Для подключения диагностического адаптера к терминалу используется интерфейс USB. После подключения, в директории /dev/ создается псевдофайл serial-порта /ttyUSB0, при помощи которого и будет происходить общение с адаптером с использованием библиотеки Pyserial. Данный модуль предоставляет бэкэнд для Python для работы с ОС Linux при помощи инкапсуляции доступа к последовательному порту.

Основная библиотека разработки данного проекта является Python-OBD. Python-OBD – это библиотека для обработки данных из порта бортовой диагностики автомобиля [19]. Она позволяет передавать данные датчиков, коды ошибок и другие важные параметры электронного блока управления. Данная библиотека предназначена для работы с адаптерами ELM327 разных чипов.

Библиотека позволяет проводить сканирование на наличие подключение serial-порта, который в свою очередь можно выставить вручную при присутствии других serial-подключений. После нахождения устройства, программа начинает открытие порта и инициализацию адаптера. После прохождения инициализации адаптера, программа приступает к

инициализации электронного блока управления автомобиля. В таблице 1 представлены протоколы, с которыми возможна работа и их идентификационный номер внутри библиотеки Python-OBD.

Таблица 1 – Доступные протоколы

ID протокола	**Названия протоколов**
1	SAE J1850 PWM
2	SAE J1850 VPW
3	ISO 9141-2
4	ISO 14230-4 (KWP 5BAUD)
5	ISO 14230-4 (KWP FAST)
6	ISO 15765-4 (CAN 11/500)
7	ISO 15765-4 (CAN 29/500)
8	ISO 15765-4 (CAN 11/250)
9	ISO 15765-4 (CAN 29/250)
A	SAE J1939 (CAN 29/250)

После прохождения инициализации блока управления, программа готова к отправке OBD-команд для получения данных датчиков, ошибок и прочих параметров электронного блока управления. Нужно отметить, что на некоторых машинах используются далеко не все OBD-команды, это связано с особенностями устройства каждого автомобиля и наличия в них специальных датчиков. Как раз для таких случаев, данная библиотека информирует о доступных OBD-командах и для удобства работы делит их на 3 группы по уровню оснащения автомобиля.

После отправки OBD-команды, программа получает на вход ответ с блока управления и расшифровывает его. Полученная информация и будет отправляться на сервер.

На рисунках 6, 7 и 8 представлены блок-схемы алгоритма работы терминала. В приложение Б приведен листинг кода терминала.

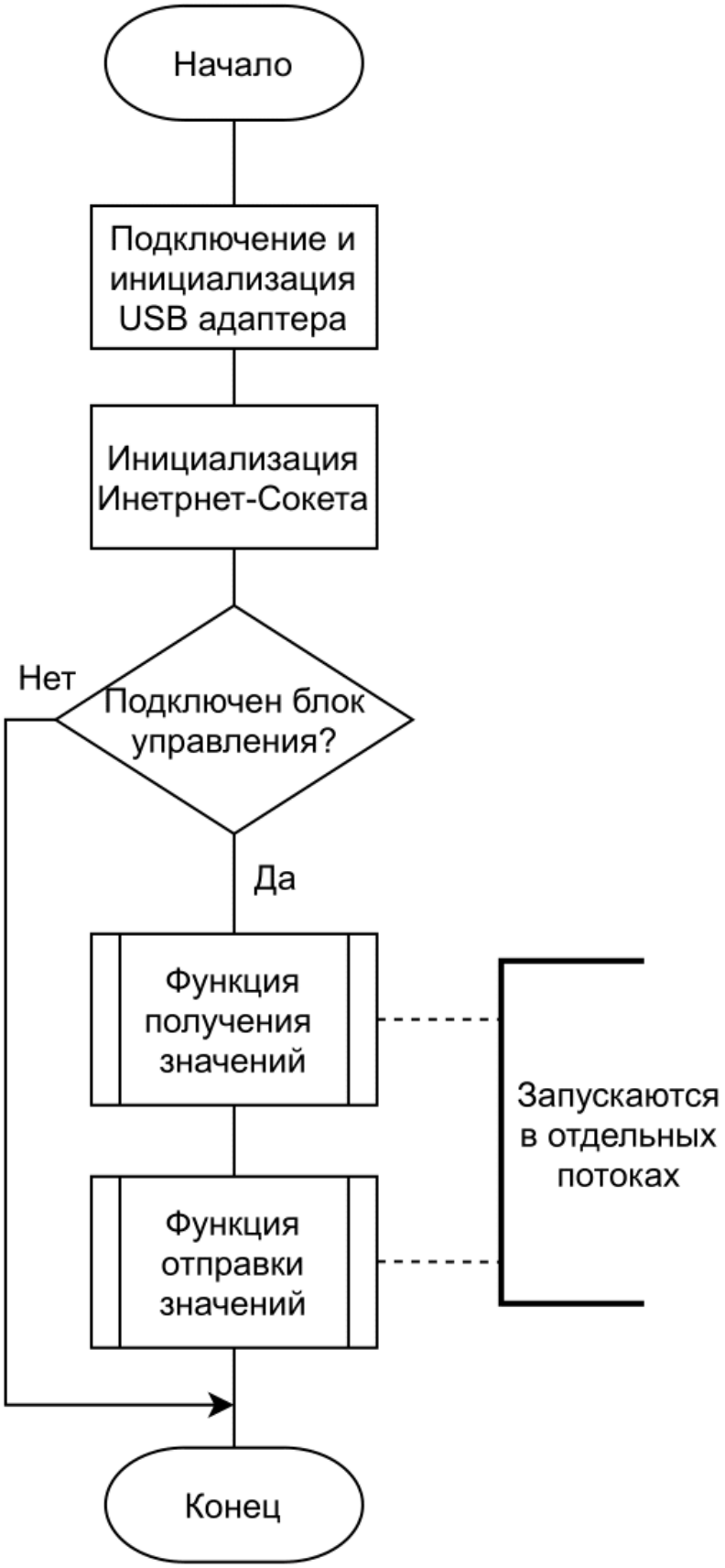

Рисунок 6 – Блок схема общего алгоритма работы терминала

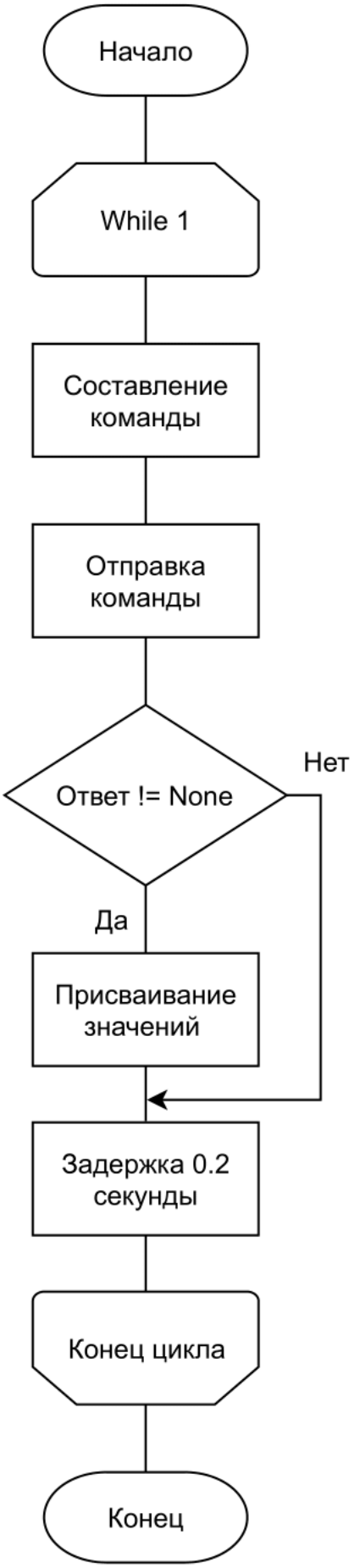

Рисунок 7 – Блок схема работы функции получения значений

Рисунок 8 – Блок схема работы функции отправки значений

3.3 Описание данных

С помощью OBD-II команд происходит получение необходимых данных из электронного блока автомобиля. В первую очередь интересуют коды ошибок и основные показатели температуры из-за того, что они являются самыми важными значениями, которые возможно получить с каждого автомобиля, в отличии, например, от данных температуры масла, датчики

которого устанавливаются не во все ТС. В таблице 2 представлены описание основных получаемых данных, которые предстоит считывать и отправлять на сервер.

Таблица 2 – Технические данные и их описание

Название	**Описание**	**Группа команды**
CURRENT_DTC	Get DTCs from the current/last driving cycle	PIDs [01-20] (1)
DTC	Get DTCs	PIDs [01-20] (1)
COOLANT_TEMP	Engine Coolant Temperature	PIDs [01-20] (1)
RUN_TIME	Engine Run Time	PIDs [01-20] (1)
FUEL_PRESSURE	Fuel Pressure	PIDs [01-20] (1)
FUEL_LEVEL	Fuel Level Input	PIDs [21-40] (2)
RPM	Vehicle Speed	PIDs [01-20] (1)
SPEED	Timing Advance	PIDs [01-20] (1)
INTAKE_PRESSURE	Intake Manifold Pressure	PIDs [01-20] (1)
INTAKE_TEMP	Intake Air Temp	PIDs [01-20] (1)

4 РЕАЛИЗАЦИЯ

4.1 Реализация аппаратной части устройства

Ниже приведены более подробные характеристики устройств, использованные в аппаратной реализации прототипа терминала.

Диагностический адаптер ELM-327 PIC18F25K80 PL2303 представлен на рисунке 9, его технические характеристики приведены в таблице 3.

Рисунок 9 – Адаптер ELM-327

Таблица 3 – Технические характеристики диагностического адаптера ELM327

Габариты	87 x 48 x 25 мм
Напряжение питания	5 В
Потребляемый ток не более	100 мА
Диапазон рабочих температур	От -25 до +40 С
Длина кабеля USB	1,7 метра

Микрокомпьютер Raspberry Pi 3 model B+ представлен на рисунке 10, его технические характеристики приведены в таблице 4.

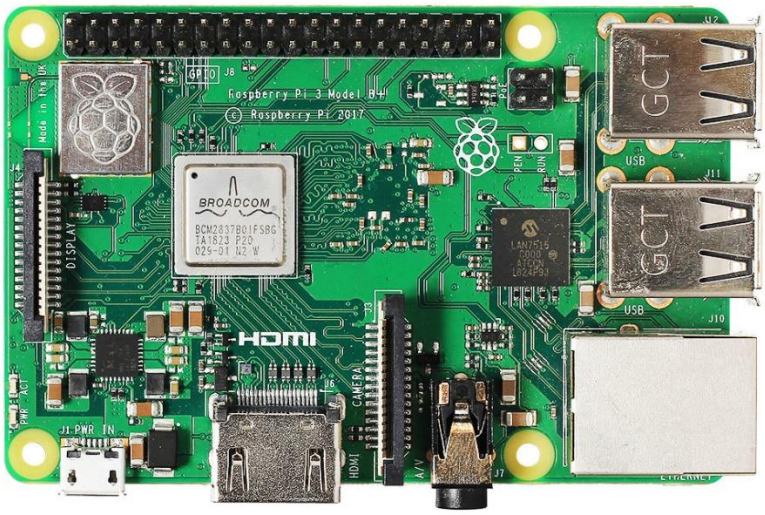

Рисунок 10 – Raspberry Pi 3 model B+

Таблица 4 – Технические характеристики Raspberry Pi 3 model B+

Процессор	ARM Cortex-A53, 1,4 ГГц
Оперативная память	1 ГБ LPDDR2 SDRAM
Цифровой видеовыход	HDMI
Композитный выход	3,5 мм (4 pin)
USB-порты	4 x USB 2.0
Беспроводная сеть	WiFi 2,4 / 5 ГГц, 802.11n
Ethernet	10/100/1000 Мб RJ45
Bluetooth	Bluetooth 4.2, BLE
Разъем дисплея	Display Serial Interface (DSI)
Разъем видеокамеры	MIPI Camera Serial Interface (CSI-2)
Карта памяти	MicroSD
Порты вывода GPIO	40
Габариты	85 x 56 x 17 мм

4.2 Реализация программной части устройства

Серверная часть реализована на языке программирования C++ с использованием библиотеки Winsock. Программа создает принимающие, содержащие в себе информацию для подключения, и связующие сокеты, которые содержат в себе сетевую информацию о подключенным к серверу терминалам. Каждый сокет рассчитан на свой вид работы – связь с терминалом или связь с программным обеспечением предприятия.

Для подключения терминалов сокет настроен на прием на 708 порту, а сокет, направленный на работу с программами предприятия может быть настроен на любой порт, но по умолчанию назначается 900 порт. Для принятия новых подключений используются методы accept_newcon() и accept_client_con(), а для их обработки уже используются методы TerminalHandle() и ClientHandle(). Методы обработки запускаются в отдельном потоке, чтобы не задерживать работу друг друга и не было проблем, связанные с очередями на обработку принятого соединения.

В качестве программирования серверной части была использована среда разработки Visual Studio. Исходный код программы серверной части комплекса приведен в приложение А.

Программная часть терминала реализована на языке Python. Также, как и на серверной части, используется технология сокетов, реализованная на данном языке при помощи модуля Socket. По аналогии с сервером, функция отправки данных выведена отдельно в поток. Исходный код программы терминала приведен в приложение Б.

Программная часть терминала, отвечающая за подключение к OBD-II интерфейсу, реализована при помощи библиотеки python-OBD. Также, как и отправка данных, их получение выведено в отдельный поток, чтобы ускорить производительность за счёт избавления от очередей среди программных операций.

Для взаимодействия терминалов, программ предприятия и сервера были разработаны специальные запросы, с помощью которых компоненты программно-аппаратного комплекса могут предавать данные между собой.

Для отправки данных от терминала до сервера используется следующий шаблон команд: $TYPE#VALUE. Здесь на месте TYPE может быть одним из типов данных. Все возможные типы данных и их описание представлены в таблице 5. На месте VALUE находится само передаваемое значение.

Для того чтобы сторонней программе получить данные, необходимо отправить команду на терминал следующего шаблона: &CODE#TYPE, где CODE – идентификационный номер, присвоенный автомобилю, а TYPE – тип передаваемых данных.

Для просмотра информации терминалов на сервера, были разработаны специальные команды, приведенные в таблице 6.

Таблица 5 – Получаемые данные терминала

Команда	**Описание**
DTC	Коды ошибок
COOLANT_TEMP	Температура охлаждающей жидкости
RPM	Обороты в минуту двигателя
SPEED	Скорость автомобиля
FUEL_LEVEL	Уровень топлива автомобиля
FUEL_PRESSURE	Давление топлива
INTAKE_PRES	Давление на впуске
INTAKE_TEMP	Температуру на впуске
RUN_TIME	Время работы автомобиля
CODE	Идентификационный код терминала

Таблица 6 – Команды консоли терминала

Команда	**Описание**
/size	Получить количество подключений
/id	Получить информацию по внутреннему индексу подключения
/value	Получить значение подключения
/all	Получить все подключения
/vall	Получить все значения
/send	Отправка команды на терминал

5 ТЕСТИРОВАНИЕ

Для проверки работоспособности программно-аппаратного комплекса было проведено альфа-тестирование.

Альфа тестирование – один из видов тестирования, в котором проверяется работоспособность продукта на поздней стадии разработки. Тестирование проводится в условиях, близким к реальным. Основной целью тестирования является получение подробной информации о готовом продукте.

5.1 Проведение процедуры тестирования

Процедура тестирования проводилась с применением автомобиля Chery Bonus A13 2011 года выпуска. На рисунке 11 представлено соединение терминала к ТС.

Для имитации работы программного обеспечения предприятия был написан простое клиент-приложение, которое может выбирать порт отправки, принимать и отправлять данные. Листинг тестовой программы представлен в приложении В.

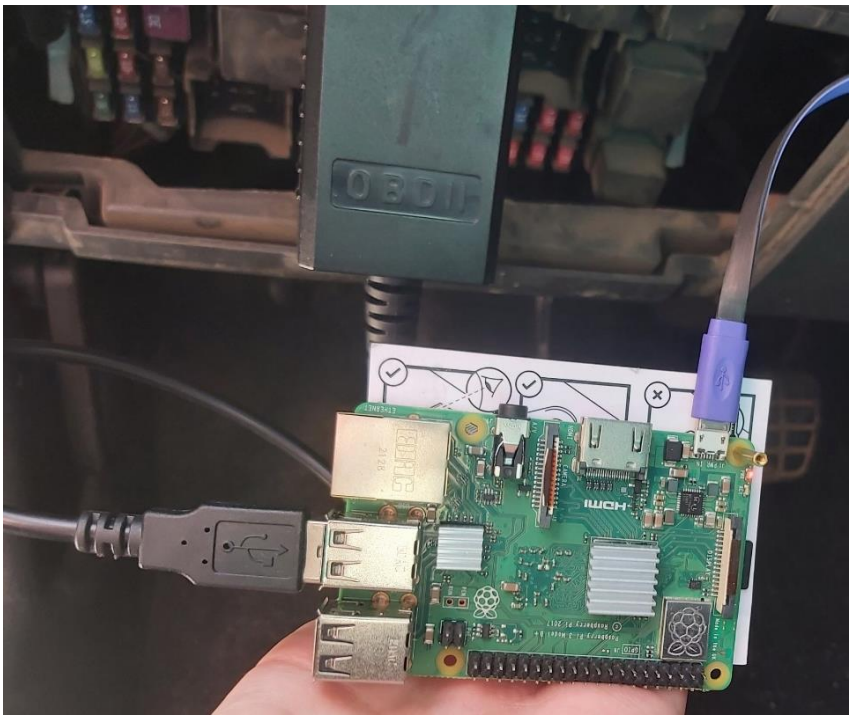

Рисунок 11 – Подключение терминала к автомобилю

Запустим сервер и выберем порт 0, который включает настройки порта по умолчанию. Затем запустим клиент и выберем порт 900, после этого подключаем терминал и запускаем программу.

Проверим подключения терминала в сервере с помощью команды /all и получение данных с него при помощи команды /vall. На рисунке 12 представлено подключение терминала к серверу и отправка технически характеристик. Также на рисунках 12 и 13 представлены параметры Fuel_pressure, Fuel_level и Run_time, которые имеют значение None. Это связано с тем, что автомобиль, на котором тестируется программно-аппаратный комплекс, не имеет возможности отправлять эти данные из-за своей особенности архитектуры взаимодействия электронных блоков.

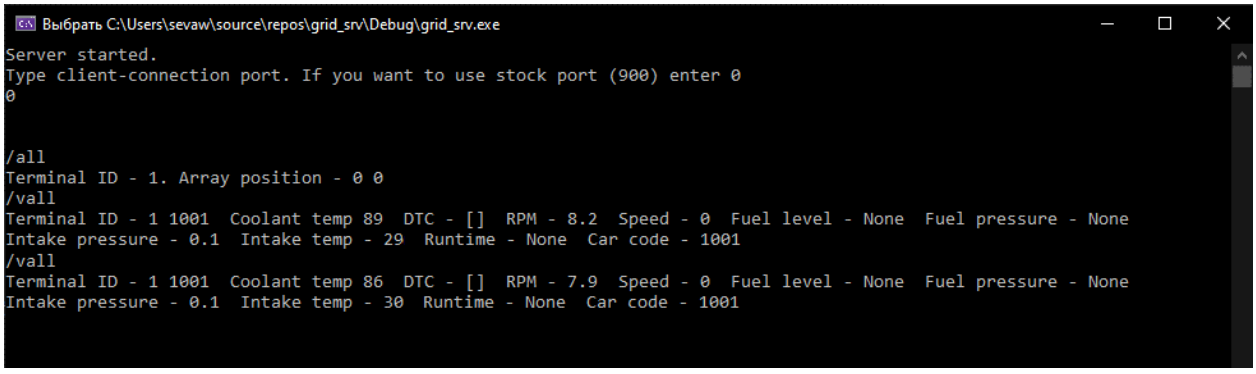

Рисунок 12 – Тестирование терминала

Далее для тестирования обработчика ошибок блока управления и их отправки, отключим датчик массового расхода воздуха, подождем несколько минут и проверим показатели.

С помощью консольной команды /vall проверяем изменение датчиков. На рисунке 13 представлена ошибка P0100 в пункте DTC и её описание. Данная ошибка свидетельствует о неисправности датчика массового расхода воздуха.

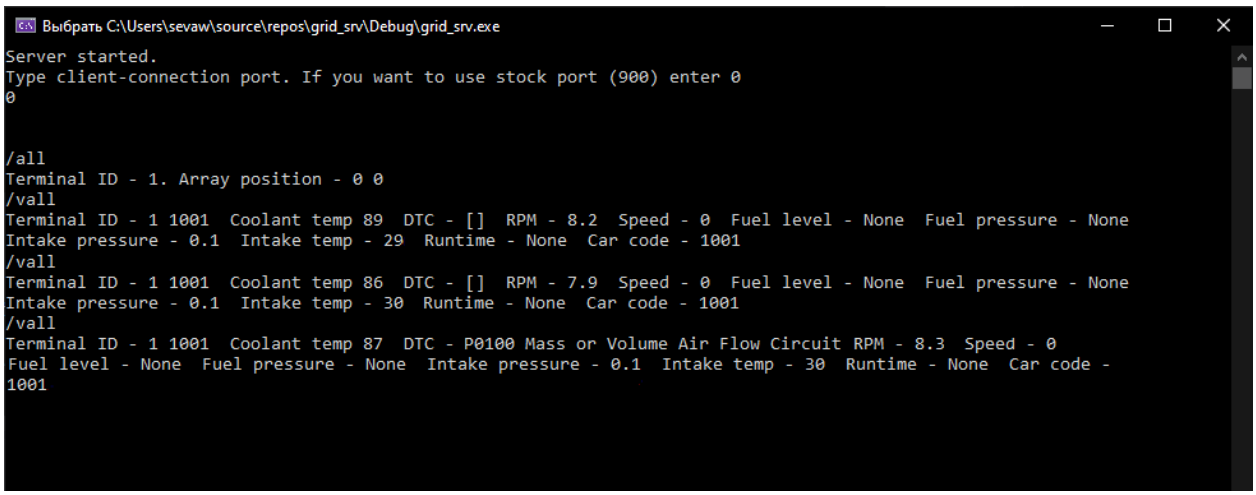

Рисунок 13 – Тестирование обработчика ошибок блока управления

Далее проверим работу программы предприятия. Для данного этапа тестирования было разработано приложение, способное только получать и отправлять данные, чтобы имитировать работу программы предприятия. Исходный код клиента представлен в приложение В. Отправим запрос с клиента на терминал и просмотрим на полученные данные.

На рисунке 14 представлена попытка отправить неверную команду и ответ сервера на неё. После того, как команда была корректно введена, нам было возвращено значение данных DTC терминала с уникальным кодом терминала 1001.

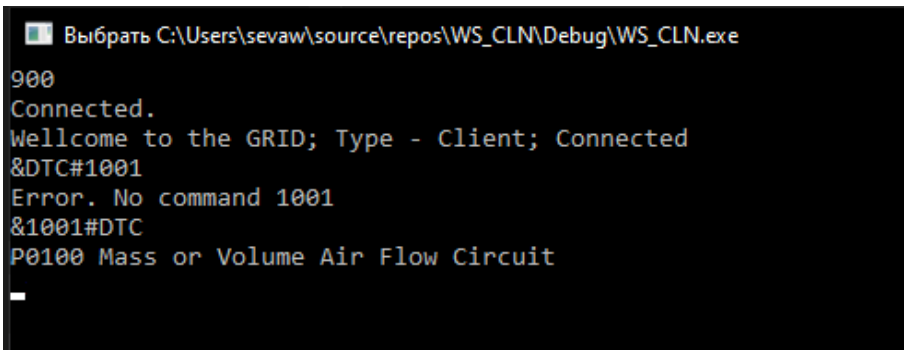

Рисунок 14 – Тестирование клиента

Попробуем проверить другие показатели. Как видно из рисунков 14 и 15, клиент может отправлять команды и получать в ответ данные технического состояния транспорта с сервера.

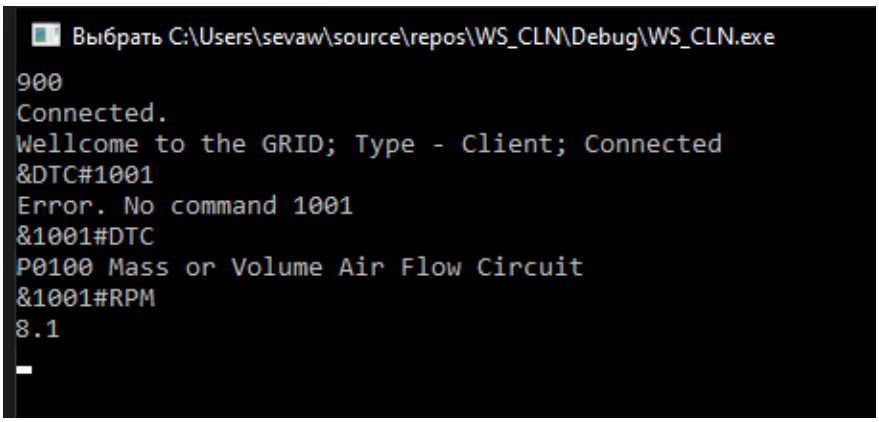

Рисунок 15 – Тестирование клиента

По итогам тестирования можно сделать вывод, что разработанный программно-аппаратный комплекс успешно работает и производит сбор данных технического состояния автомобилей.

ЗАКЛЮЧЕНИЕ

В представленной работе были проработаны все этапы, связанные с проектированием и разработкой программно-аппаратного комплекса.

Был проведен обзор аналогичных решений на внутреннем рынке страны, который показал отсутствие большого количества решений, а из доступных не все предоставляют возможность открытого внедрения в уже созданные системы предприятий.

Были рассмотрены технические решения программного комплекса, результатом которого стал выбор диагностического адаптера ELM327 с чипом PIC18F25K80 PL2303 и микрокомпьютера Raspberry Pi 3 model B+.

Был разработан программно-аппаратный комплекс, позволяющий собирать данные технического состояния автомобиля удаленно в режиме реального времени.

Производилось альфа-тестирование проекта, результатом тестирования можно объявить то, что система работает стабильно и без нареканий, выполняет поставленные ранее задачи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Байбакова, А. А. Техническая неисправность автомобиля как одна из причин ДТП / А. А. Байбакова // Автомобильный транспорт Дальнего Востока. – 2018. – № 1. – С. 31-34.

2. История OBD I и OBD II. – Текст. Изображение (неподвижное ; двумерное) : электронные // Автомодерн : [сайт]. – 2022. – URL: https://ustroistvo-avtomobilya.ru/sistemy-snizheniya-toksichnosti/istoriya-obd-i-i-obd-ii/ (дата обращения 13.05.2022).

3. Что такое OBD II. – Текст. Изображение (неподвижное ; двумерное) : электронные // Автосканеры.ru : [сайт]. – 2022. – URL: https://www.autoscaners.ru/articles/what-is-obd-ii/ (дата обращения 13.05.2022).

4. Что такое телематика и зачем она вам нужна. – Текст. Изображение (неподвижное ; двумерное) : электронные // Лаборатория Умного Вождения : [сайт]. – 2021. – URL: https://smartdriving.io/blog/chto-takoe-telematika-i-zachem-ona-vam-nuzhna/ (дата обращения 15.05.2022).

5. Телематика для автомобиля. – Текст : электронный // Ассоциация международных автомобильных перевозчиков : [сайт]. – 2022. – URL: http://bamap.org/information/smi/2021_02_01_138747/print/ (дата обращения 16.05.2022).

6. Система Geostron. – Текст : [официальный сайт] / Geostron. – Россия, 2022 – . – URL: https://geostron.ru/ (дата обращения: 16.05.2022). – Текст. Изображение : электронные.

7. Автомобильный трекер Teltonika FMB 010. – Текст. Изображение (неподвижное ; двумерное) : электронные // Geostron : [сайт]. – 2022. – URL: https://geostron.ru/ru/katalog/trackers/avtomobilnyj-treker-teltonika-fmb-010/ (дата обращения 16.05.2022).

8. Автомобильный трекер Teltonika FMB 002. – Текст. Изображение (неподвижное ; двумерное) : электронные // Geostron : [сайт]. – 2022. – URL:

https://geostron.ru/ru/katalog/trackers/avtomobilnyj-treker-teltonika-fmb-003/ (дата обращения 16.05.2022).

9. Оборудование для мониторинга автотранспорта. – Текст. Изображение (неподвижное ; двумерное) : электронные // TrackGPS : [сайт]. – 2022. – URL: https://www.trackgps.ru/gps-treker-dlya-avto.html (дата обращения 16.05.2022).

10. Montrans : [официальный сайт] / Montrans. – Россия, 2022 – . – URL: https://montrans.ru/ (дата обращения: 16.05.2022). – Текст. Изображение: электронные.

11. Cartrek : [официальный сайт] / Cartrek. – Россия, 2022 – . – URL: https://cartrek.online/ru/ (дата обращения: 16.05.2022). – Текст. Изображение: электронные.

12. Общие сведения о программировании на C++ в Windows. – Текст. Изображение (неподвижное ; двумерное) : электронные // Microsoft : [сайт]. – 2022. – URL: https://docs.microsoft.com/ru-ru/cpp/windows/overview-of-windows-programming-in-cpp?view=msvc-170 (дата обращения 18.05.2022).

13. Что такое C#. – Текст. Изображение (неподвижное ; двумерное) : электронные // Web-Proger : [сайт]. – 2022. – URL: http://web.spt42.ru/index.php/chto-takoe-c (дата обращения 18.05.2022).

14. Альтернативы Raspberry Pi. – Текст. Изображение (неподвижное ; двумерное) : электронные // Habr: ресурс для IT-специалистов : [сайт]. – 2022. – URL: https://habr.com/ru/post/457666/ (дата обращения 22.05.2022).

15. Сычев, Д. Отличия ELM327 адаптеров разных версий и чипов / Д. Сычёв. – Текст. Изображение (неподвижное ; двумерное) : электронные // Я Диагност : [сайт]. – 2022. – URL: https://yadiagnost.ru/elm327-otlichiya (дата обращения 22.05.2022).

16. Отличия адаптеров ELM327 USB на чипе FTDI, Prolific и CH340. – Текст. Изображение (неподвижное ; двумерное) : электронные // Drive2.RU : [сайт]. – 2022. – URL: https://www.drive2.ru/b/522535205592892168/ (дата обращения 22.05.2022).

17. Принципы сокетов. – Текст. Изображение (неподвижное ; двумерное) : электронные // Лекции кафедры ИИТ УрФУ: [сайт]. – 2022. – URL: https://lecturesnet.readthedocs.io/net/low-level/ipc/socket/intro.html (дата обращения 22.05.2022).

18. О Winsock. – Текст. Изображение (неподвижное ; двумерное) : электронные // Microsoft : [сайт]. – 2022. – URL: https://docs.microsoft.com/ru-ru/windows/win32/winsock/about-winsock (дата обращения 23.05.2022).

19. Whitfield, B. Python-OBD / B. Whitfield. – Текст. Изображение (неподвижное ; двумерное) : электронные // Python-OBD Documentation : [сайт]. – 2022. – URL: https://python-obd.readthedocs.io/en/latest/ (дата обращения 23.05.2022).

ПРИЛОЖЕНИЕ А

Программа сервера

Листинг А.1 – Исходный код сервера

```
#pragma comment(lib, "ws2_32.lib")
#include <winsock2.h>
#include <iostream>
#include <thread>
#include <vector>
#include <string>
#include <map>

#pragma warning(disable: 4996)

class clt_cnt
{

private:

	SOCKET prodcl;

	struct terminal
	{
		int id;
		SOCKET client;

		std::string
			DTC = "None",
			COOLANT_TEMP = "None",
			RPM = "None",
			SPEED = "None",
			FUEL_LEVEL = "None",
			FUEL_PRESSURE = "None",
			INTAKE_PRES = "None",
			INTAKE_TEMP = "None",
			RUN_TIME = "None",
			CODE = "0";
	};
	std::vector<terminal> connections;
	std::map <std::string, int> mapping;
public:

	clt_cnt()
	{
		mapping["DTC"] = 1;
		mapping["COOLANT_TEMP"] = 2;
		mapping["RPM"] = 3;
		mapping["SPEED"] = 4;
		mapping["FUEL_LEVEL"] = 5;
		mapping["FUEL_PRESSURE"] = 6;
		mapping["INTAKE_PRES"] = 7;
		mapping["INTAKE_TEMP"] = 8;
		mapping["RUN_TIME"] = 9;
		mapping["CODE"] = 10;
		mapping["ALL"] = 11;
	}

	void set_value(int id, std::string type, std::string value)
```

```
	{
		int index = search(id);

		switch (mapping[type])
		{
			case 1: connections[index].DTC = value; break;
			case 2: connections[index].COOLANT_TEMP = value; break;
			case 3: connections[index].RPM = value; break;
			case 4: connections[index].SPEED = value; break;
			case 5: connections[index].FUEL_LEVEL = value; break;
			case 6: connections[index].FUEL_PRESSURE = value; break;
			case 7: connections[index].INTAKE_PRES = value; break;
			case 8: connections[index].INTAKE_TEMP = value; break;
			case 9: connections[index].RUN_TIME = value; break;
			case 10: connections[index].CODE = value; break;
			default: break;
		};

	}

	void get_value(int id, std::string type)
	{
		int index = search(id);

		switch (mapping[type])
		{
		case 1: std::cout << connections[index].DTC; break;
		case 2: std::cout << connections[index].COOLANT_TEMP<<std::endl;
break;
		case 3: std::cout << connections[index].RPM << std::endl; break;
		case 4: std::cout << connections[index].SPEED << std::endl; break;
		case 5: std::cout << connections[index].FUEL_LEVEL << std::endl;
break;
		case 6: std::cout << connections[index].FUEL_PRESSURE <<
std::endl; break;
		case 7: std::cout << connections[index].INTAKE_PRES << std::endl;
break;
		case 8: std::cout << connections[index].INTAKE_TEMP << std::endl;
break;
		case 9: std::cout << connections[index].RUN_TIME << std::endl;
break;
		case 10: std::cout << connections[index].CODE << std::endl; break;
		case 11: std::cout
			<< " Coolant temp " << connections[index].COOLANT_TEMP << "
"
			<< " DTC - " << connections[index].DTC << " "
			<< " RPM - " << connections[index].RPM << " "
			<< " Speed - " << connections[index].SPEED << " "
			<< " Fuel level - " << connections[index].FUEL_LEVEL << " "
			<< " Fuel pressure - " << connections[index].FUEL_PRESSURE
<< " "
	<< " Intake pressure - " << connections[index].INTAKE_PRES << " "
	<< " Intake temp - " << connections[index].INTAKE_TEMP << " "
	<< " Runtime - " << connections[index].RUN_TIME << " "
	<< " Car code - " << connections[index].CODE << " "
	<< std::endl; break;
		default: break;
		};
	}

	std::string get_code_value(std::string code, std::string value)
```

```
	{
		int index = searc_code(code);
		std::string none = "Error. No command " + value;

		switch (mapping[value])
		{
			case 1: return connections[index].DTC; break;
			case 2: return connections[index].COOLANT_TEMP; break;
			case 3: return connections[index].RPM; break;
			case 4: return connections[index].SPEED; break;
			case 5: return connections[index].FUEL_LEVEL; break;
			case 6: return connections[index].FUEL_PRESSURE; break;
			case 7: return connections[index].INTAKE_PRES; break;
			case 8: return connections[index].INTAKE_TEMP; break;
			case 9: return connections[index].RUN_TIME; break;
			case 10: return connections[index].CODE; break;
			default: return none; break;
		};

	}

	void get_all_term()
	{
		for (int ch = 0; ch < connections.size(); ch++) {

std::cout << "Terminal ID - " << connections[ch].id << ". Array position - "
<< ch << " " << connections[ch].CODE <<std::endl;
		}

	}
	void get_all_val()
	{
		std::string atr = "ALL";
		for (int cvh = 0; cvh < connections.size(); cvh++) {
			std::cout << "Terminal ID - " << connections[cvh].id << " "
<< connections[cvh].CODE << " ";
			get_value(connections[cvh].id, atr);
		}
	}

	int search(int index)
	{
		for (int cs = 0; cs < connections.size(); cs++)
			if (connections[cs].id == index) return cs;
	}

	int searc_code(std::string code)
	{
		for (int cs = 0; cs < connections.size(); cs++)
			if (connections[cs].CODE == code) return cs;
	}

	bool isok(int index)
	{
		for (int cs = 0; cs < connections.size(); cs++)
			if (connections[cs].id == index) return 1;
		return 0;
	}

	void add_terminal(terminal trm)
	{
```

```
	connections.push_back(trm);
	}
	int get_size()
	{
		return connections.size();
	}

	void get_id()
	{
		std::cout << "id? - ";
		int i; std::cin >> i;
		if (i <= get_size()) std::cout << "\n INDEX - " << i << " on adr -
" << search(i) << std::endl;
		else std::cout << "Out of range\n";
	}

	void get_id(int i)
	{
		std::cout << "\n INDEX - " << i << " on adr - " << search(i) <<
std::endl;
	}

	void TerminalCommands(std::string command, int id)
	{
		std::map <std::string, int> mapping;
		mapping["/Techerror"] = 1;
		mapping["/ID"] = 2;
		switch (mapping[command]) {
			case 1: std::cout << " Technical error on ID -" << id <<
std::endl; break;
			case 2: get_id(id); break;
		};
	}

	void ClientCommands(std::string command)
	{
		std::string code = command.substr(command.find("&") + 1,
	command.find("#") - 1);
		std::string value = command.substr(command.find("#") + 1);
		std::cout << code << " " << value << std::endl;
		std::string tmp = get_code_value(code, value);
		std::cout << tmp << std::endl;

		char msg[256];
		strcpy(msg, tmp.c_str());
		send(prodcl, msg, sizeof(msg), NULL);

	}

	void TerminalHandle(int index)
	{
		char msg[256];
		std::string instr;

		while (1) {
			if (SOCKET_ERROR == recv(connections[search(index)].client,
msg, sizeof(msg), NULL)) {
				std::cout << index << " terminal. Error receiving " <<
WSAGetLastError() << std::endl;
				break;
			}
```

```
            instr = std::string(msg);

                  if (msg[0] == '/') TerminalCommands(instr, index);
                        else
                  if (msg[0] == '$')      set_value(index,
instr.substr(instr.find("$") + 1, instr.find("#") - 1),
instr.substr(instr.find("#") + 1));
                  std::cout << msg << std::endl;
            }
            connections.erase(connections.begin() + search(index));
      };

      void ClientHandle()
      {
            char msg[256];
      std::string instr;

            while (1) {
      if (SOCKET_ERROR == recv(prodcl, msg, sizeof(msg), NULL)) {
                        std::cout << " Client. Error receiving " <<
WSAGetLastError() << std::endl;
                        break;
                  }
                  instr = std::string(msg);
                  //std::cout << msg << std::endl;
                  if (msg[0] == '&') ClientCommands(instr);
                  }
      };

      void accept_newcon(SOCKET sListen, SOCKADDR_IN addr, int id)
      {
            int addr_ = sizeof(addr);
            SOCKET newConnection = accept(sListen, (SOCKADDR*)&addr, &addr_);
            terminal* temp_connection = new terminal{ id, newConnection };
            if (!newConnection) {
                  std::cout << "Error\n";
            } else {
                  char msg[256] = "Wellcome to the GRID; Type - Terminal;
Connected with ID - ";
                  sprintf(msg + strlen(msg), " %d ", id);
                  send(newConnection, msg, sizeof(msg), NULL);
                  add_terminal(*temp_connection);
            }
            delete temp_connection;
      }

      void accept_client_con(SOCKET sListen, SOCKADDR_IN addr)
      {
            int addr_ = sizeof(addr);
            SOCKET newConnection = accept(sListen, (SOCKADDR*)&addr, &addr_);
            prodcl = newConnection;
      if (!newConnection) {
                  std::cout << "Error\n";
            }
            else {
                  char msg[256] = "Wellcome to the GRID; Type - Client;
Connected ";
                  send(newConnection, msg, sizeof(msg), NULL);
            }
      }
```

```
void send_comm(int index, std::string cmd)
	{
		char ms[24];
		strcpy(ms, cmd.c_str());
		send(connections[search(index)].client, ms, sizeof(ms), NULL);
	}
};

clt_cnt Terminals;

void CTH(int i)
{
	Terminals.TerminalHandle(i);
}

void CCH()
{
	Terminals.ClientHandle();
}

void ConsoleHandle()
{
	std::map <std::string, int> consolecommands;
	consolecommands["/size"]	= 1;
	consolecommands["/id"]		= 2;
	consolecommands["/value"]	= 3;
	consolecommands["/all"]		= 4;
	consolecommands["/vall"]	= 5;
	consolecommands["/send"]	= 6;


	int tmpint;
	std::string tmpstr;
	std::string str;

	while (true) {

		std::cin >> str;
		switch (consolecommands[str]) {
			case 1: std::cout<<Terminals.get_size()<<std::endl; break;
			case 2: Terminals.get_id(); break;
			case 3: {
				std::cout << "Input index - ";
				std::cin >> tmpint;
				std::cout << "Input value - ";
				std::cin >> tmpstr;
				if (Terminals.isok(tmpint))
Terminals.get_value(tmpint, tmpstr);
				else std::cout << "Bad";
				break;
			}
			case 4: Terminals.get_all_term(); break;
			case 5: Terminals.get_all_val(); break;
			case 6: {
				std::cout << "Input index - ";
				std::cin >> tmpint;
				std::cout << "Input command - ";
				std::cin >> tmpstr;
			if (Terminals.isok(tmpint))Terminals.send_comm(tmpint, tmpstr);
```

```
		else std::cout << "Bad";
                        break;
                    }
                }


        }
}
int main(int argc, char* argv[]) {
        WSAData wsaData;
        WORD DLLVersion = MAKEWORD(1, 2);
        if (WSAStartup(DLLVersion, &wsaData) != 0) {
                std::cout << "Error. Server is not running \n";
        }
        else std::cout << "Server started. \n";

        ///*** TERMINAL NETWORK SETTINGS ***///

        SOCKADDR_IN addr;
        int addr_ = sizeof(addr);
        addr.sin_addr.s_addr = htonl(INADDR_ANY); //Прослушка на любой адресс,
расчитываем на прямую видимость; заменен с inet_addr("127.0.0.1");
        addr.sin_port = htons(708);
        addr.sin_family = AF_INET;

        SOCKET sListen = socket(AF_INET, SOCK_STREAM, NULL);
        bind(sListen, (SOCKADDR*)&addr, sizeof(addr));
        listen(sListen, SOMAXCONN);

        ///*** CLIENT NETWORK SETTINGS ***///

        int port;
        std::cout << "Type client-connection port. If you want to use stock port
(900) enter 0\n";
        std::cin >> port;
        if (!port) port = 900;

        SOCKADDR_IN claddr;
        int claddr_ = sizeof(claddr);
        claddr.sin_addr.s_addr = htonl(INADDR_ANY); //Прослушка на любой адресс,
расчитываем на прямую видимость; заменен с inet_addr("127.0.0.1");
        claddr.sin_port = htons(port);
        claddr.sin_family = AF_INET;

        SOCKET clListen = socket(AF_INET, SOCK_STREAM, NULL);
        bind(clListen, (SOCKADDR*)&claddr, sizeof(claddr));
        listen(clListen, SOMAXCONN);

        ///*** ***///

        CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ConsoleHandle, NULL,
NULL, NULL);

        SOCKET newConnetcion;

        Terminals.accept_client_con(clListen, claddr);
        CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)CCH, NULL, NULL, NULL);

        int i = 1;
        while (true) {
                Terminals.accept_newcon(sListen, addr, i);
```

```
	CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)CTH, (LPVOID)(i), NULL,
NULL);
		i++;
	}

	return 0;
}
```

ПРИЛОЖЕНИЕ Б

Программа терминала

Листинг Б.1 – Исходный код терминала

```
#!/usr/bin/python
from __future__ import division
# -*- coding: utf-8 -*-

import threading
import time
import sys
import obd
import socket

clientsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

CODE="0"
GET_RPM="0"
GET_SPEED="0"
GET_DTC="0"
GET_COOLANT_TEMP="0"
GET_RUN_TIME="0"
GET_INTAKE_TEMP="0"
GET_INTAKE_PRES="0"
GET_FUEL_STATUS=" "
GET_FUEL_PRESSURE="0"

def str_int(str,digits=0):
    index=str.find(".")
#    print index
    if index==-1:
      return str
    if index>0:
      if digits>0:
          return str[0:index+digits+1]
      return str[0:idex]
   return "0"
def getValues():
    global
GET_RPM,GET_SPEED,GET_DTC,GET_COOLANT_TEMP,GET_RUN_TIME,GET_INTAKE_TEMP,GET_I
NTAKE_PRES,GET_FUEL_STATUS,GET_FUEL_PRESSURE,CODE

      CODE = ("$SET_CODE#" + str(CODE))
      clientsocket.send(str.encode(CODE))
      time.sleep(2)

    while True:
      cmd = obd.commands.RPM
      response = connection.query(cmd)
      if response.value!=None:
          GET_RPM=( "$RPM#"+ str(response.value))

      clientsocket.send(str.encode(RPM))
      time.sleep(2)

      cmd = obd.commands.SPEED
      response = connection.query(cmd)
      if response.value!=None:
```

```
            GET_SPEED=( "$SPEED#"+ str(response.value))

        clientsocket.send(str.encode(SPEED))
        time.sleep(2)

        cmd = obd.commands.COOLANT_TEMP
        response = connection.query(cmd)
        if response.value!=None:
            GET_COOLANT_TEMP=( "$COOLANT_TEMP#"+ str(response.value))

        clientsocket.send(str.encode(COOLANT_TEMP))
        time.sleep(2)
        clientsocket.send(str.encode(COOLANT_TEMP))
        cmd = obd.commands.RUN_TIME
        response = connection.query(cmd)
        if response.value!=None:
            GET_RPM=( "$RUN_TIME#"+ str(response.value))
        clientsocket.send(str.encode(RUN_TIME))
        time.sleep(2)

        cmd = obd.commands.INTAKE_PRES
        response = connection.query(cmd)
        if response.value!=None:
            GET_RPM=( "$INTAKE_PRES#"+ str(response.value))

        clientsocket.send(str.encode(INTAKE_PRES))
        time.sleep(2)

        cmd = obd.commands.INTAKE_TEMP
        response = connection.query(cmd)
        if response.value!=None:
            GET_RPM=( "$INTAKE_TEMP#"+ str(response.value))

        clientsocket.send(str.encode(INTAKE_TEMP))
        time.sleep(2)

        cmd = obd.commands.FUEL_STATUS
        response = connection.query(cmd)
        if response.value!=None:
            GET_RPM=( "$FUEL_STATUS#"+ str(response.value))

        clientsocket.send(str.encode(FUEL_STATUS))
        time.sleep(2)

        cmd = obd.commands.FUEL_PRESSURE
        response = connection.query(cmd)
        if response.value!=None:
            GET_RPM=( "$FUEL_PRESSURE#"+ str(response.value))


        clientsocket.send(str.encode(FUEL_PRESSURE))
        time.sleep(2)

########################################################################

connection = obd.OBD("/dev/ttyUSB0") # auto-connects to USB or RF port

SERVER = str(input("IP:"))
PORT = int(input("Port:"))
clientsocket.connect((SERVER, PORT))
```

```
CODE = int(input("Code:"))

test = obd.commands.RPM
response = connection.query(cmd)
if response.value!=None
      Thread_getValues=threading.Thread(target=getValues)
      Thread_getValues.daemon = False
      Thread_getValues.start()
```

ПРИЛОЖЕНИЕ В

Программа клиента

Листинг В.1 – Исходный код тестового клиента

```
//#include "stdafx.h"
#pragma comment(lib, "ws2_32.lib")
#include <winsock2.h>
#include <iostream>

#pragma warning(disable: 4996)

SOCKET Connection;

void ClientHandler() {
	char msg[256];
	while (true) {
		if (SOCKET_ERROR == recv(Connection, msg, sizeof(msg), NULL)) {
			std::cout << "Error receiving " << WSAGetLastError() << std::endl;
			break;
		}
		std::cout << msg << std::endl;
	}
}

int main(int argc, char* argv[]) {
	//WSAStartup
	WSAData wsaData;
	//WORD DLLVersion = MAKEWORD(2, 1);
	if (WSAStartup(MAKEWORD(2, 1), &wsaData) != 0) {
		std::cout << "Error" << std::endl;
		exit(1);
	}

	int temp = 0;
	std::cin >> temp;

	SOCKADDR_IN addr;
	addr.sin_addr.s_addr = inet_addr("127.0.0.1");
	addr.sin_port = htons(temp);
	addr.sin_family = AF_INET;

	Connection = socket(AF_INET, SOCK_STREAM, NULL);
	if (connect(Connection, (SOCKADDR*)&addr, sizeof(addr)) == SOCKET_ERROR) {
		std::cout << "Error: failed connect to server. WSA last error:  " <<
WSAGetLastError() << std::endl;
		return 1;
	}
	std::cout << "Connected.\n";

	char msg[256];
	recv(Connection, msg, sizeof(msg), NULL);
	std::cout << msg << std::endl;

	CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)ClientHandler, NULL, NULL, NULL);

	char msgdta[256];

	while (true) {
		std::cin.getline(msgdta, sizeof(msgdta));
```

```
        if (msgdta[0] == '^') {
                        exit(1);
                }
send(Connection, msgdta, sizeof(msgdta), NULL);
                Sleep(10);
        }

        closesocket(Connection);

        system("pause");
        return 0;
        }
```