«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	11.03.01 – Радиотехника
Профиль	Радиоэлектронные системы
Факультет	РТ
Кафедра	РС

К защите допустить

Зав. кафедрой _____

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: «Сетевое взаимодействие в клиент-серверной архитектуре микро-контроллер - персональный компьютер»

Студент		_____ *подпись*	Бектемиров М.Р.
Руководитель	к.т.н., доцент каф. РС *(Уч. степень, уч. звание)*	_____ *подпись*	Смирнов Б.И.
Консультанты	асс. каф. РЭС СПбГЭТУ *(Уч. степень, уч. звание)*	_____ *подпись*	Проценко И.М.
	к.т.н., доц. *(Уч. степень, уч. звание)*	_____ *подпись*	Иванов А.Н.
	к.т.н., доц. *(Уч. степень, уч. звание)*	_____ *подпись*	Маругин А.С.

Санкт-Петербург

2023

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю

Зав. кафедрой РС

_____

«___»_____2023 г.

Студент Бектемиров Макар Русланович Группа 9101

Тема работы: «Сетевое взаимодействие в клиент-серверной архитектуре микроконтроллер - персональный компьютер»

Место выполнения ВКР: СПбГЭТУ «ЛЭТИ»

Исходные данные (технические требования):

Разработка программы для управления макетом под руководством МК, а также передачи данных между МК и ПК в сети Wi-Fi. Взаимодействие между субъектами происходит по RESTless протоколу в виде JSON пакетов с конфигурационными данными установленного образца. Источники информации на стороне макета – сенсоры, приемники информации: светодиодные ленты и дискретные светодиоды.

Содержание ВКР: описание принципов сетевой архитектуры клиент-сервер, реализация взаимодействия между МК и ПК в беспроводной сети Wi-Fi.

Перечень отчетных материалов: пояснительная записка, иллюстративный материал, презентация в среде PowerPoint

Дополнительные разделы: Безопасность жизнедеятельности

Дата выдачи задания	Дата представления ВКР к защите
«15» марта 2023 г.	«____» _____2023 г.

Студент _____ Бектемиров М.Р.

Руководитель к.т.н., доцент каф. РС _____ Смирнов Б.И.

(Уч. степень, уч. звание) *подпись*

Консультант асс. каф. РЭС СПбГЭТУ _____ Проценко И.М.

(Уч. степень, уч. звание) *подпись*

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю

Зав. кафедрой РС

_____

«___»_____2023 г.

Студент Бектемиров М.Р. Группа 9101

Тема работы: «Сетевое взаимодействие в клиент-серверной архитектуре микроконтроллер - персональный компьютер»

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	20.03 – 26.03
2	Сетевая архитектура клиент-сервер	27.03 – 31.03
3	Интерфейс сокетов	01.04 – 06.04
4	Описание устройства макета	07.04 – 11.04
5	Разработка серверной программы для управления МК	12.04 – 26.04
6	Разработка клиентской программы для ПК	27.04 – 03.05
7	Безопасность жизнедеятельности	04.05 – 11.05
8	Оформление пояснительной записки	10.05 – 17.05
9	Оформление иллюстративного материала	18.05 – 30.05

Студент _____ Бектемиров М.Р.
подпись

Руководитель к.т.н., доцент каф. РС _____ Смирнов Б.И.
(Уч. степень, уч. звание) *подпись*

Консультант асс. каф. РЭС СПбГЭТУ _____ Проценко И.М.
(Уч. степень, уч. звание) *подпись*

РЕФЕРАТ

Пояснительная записка

Ключевые слова и словосочетания: сетевая архитектура, клиент, сервер, сокет, разработка, Wi-Fi, Arduino, JSON, Python.

Объектом разработки является программа для управления микроконтроллером в клиент-серверной архитектуре микроконтроллер-персональный компьютер.

Цель работы – исследование принципов взаимодействия субъектов в сетевой архитектуре «клиент-сервер», реализация такой архитектуры в беспроводной сети Wi-Fi с применением текстового формата обмена данными JSON в системе МК-ПК.

В данной работе проведено исследование принципов взаимодействия субъектов в сетевой архитектуре «клиент-сервер», рассмотрены возможности организации такой архитектуры в беспроводной сети Wi-Fi с применением текстового формата обмена данными JSON. Разработана программа для управления микроконтроллером в описанной архитектуре, а также программа для приема и обработки полученных данных на персональном компьютере.

ABSTRACT

The aim of the work is to study the principles of interaction between subjects in the network architecture "client-server", to consider the possibility of organizing this architecture in a wireless Wi-Fi network using the JSON text data exchange format.

In this study, a program was developed for controlling a microcontroller in the described architecture, as well as a program for receiving and processing the received data on a personal computer.

СОДЕРЖАНИЕ

Введение

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

АЦП – аналого-цифровой преобразователь

БД – базы данных

МК – микроконтроллер

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

СУБД - система управления базами данными

AP – access point

API – application programming interface

ASCII – American standard code for information interchange

GPIO – general-purpose input/output

GUI – graphic user interface

I2C - Inter-Integrated Circuit

IDE – Integrated Development Environment

IOT – internet of things

IP – internet protocol

JSON - JavaScript Object Notation

SCL – serial clock

SDA – serial data

ST – station

TCP – transmission control protocol

UDP – user datagram protocol

ВВЕДЕНИЕ

В настоящее время всё большее количество электронных устройств и вещей, окружающих человека в повседневной жизни, вовлекается в концепцию интернета вещей (IOT), согласно которой между ними строится система передачи данных, призванная оптимизировать их работу в аспектах взаимодействия с человеком [1]. Так, например, дистанционное управление бытовыми приборами экономит немало времени при выполнении ежедневных домашних обязанностей, или дистанционный сбор информации с различных устройств и автоматический её анализ могут послужить основой для исследований или оптимизации любых аспектов жизни человека, будь то те же домашние обязанности или профессиональная деятельность.

Концепция IOT с момента своего появления наполняется новым технологическим и практическим содержанием в связи с устойчивым развитием информационных технологий, распространением беспроводных сетей и их совершенствованием, а также появлением облачных вычислений.

Главными субъектами во взаимодействиях в рамках описанной концепции являются устройства, которые представляют из себя приемники или источники информации, и человек, который управляет устройствами и интерпретирует полученные от них данные.

Рассматриваемое взаимодействие может быть описано и реализовано с помощью сетевой архитектуры клиент-сервер, в которой роль сервера выполняет устройство, включающее в себя источники и приемники информации, а роль клиента – приложение для управления этим устройством.

Итак, целью данной работы является разработка программы управления макетом (сервером) с приемниками и источниками информации, руководимым МК, а также программного обеспечения для персонального компьютера (клиента), цель которого – управление макетом. При этом взаимодействие между клиентом и сервером осуществляется с помощью беспроводной сети Wi-Fi.

1. СЕТЕВАЯ АРХИТЕКТУРА КЛИЕНТ-СЕРВЕР

1.1. Классическая двухуровневая архитектура клиент-сервер

Клиент-сервер – это сетевая архитектура, в которой нагрузка распределена между поставщиком данных, называемым сервером, и получателями данных, называемыми клиентами [2]. При этом приложение, являющееся клиентом, выполняет активную функцию инициации соединения и выполнения запросов, а приложение-сервер пассивно на них отвечает. Во время работы сервер может выполнять функции клиента, запрашивая и получая от него данные, и наоборот. Пример двухуровневой системы клиент-сервер представлен на рис.1.

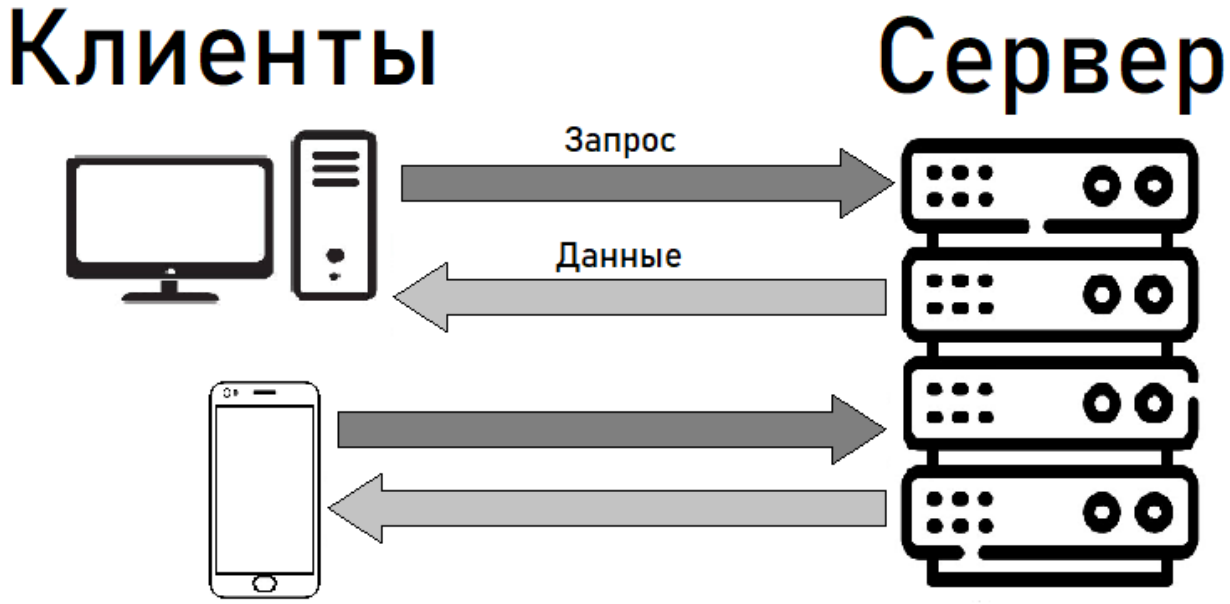


Рисунок 1 – Классическая двухуровневая архитектура клиент-сервер

Любая информационная система имеет три основные части: хранение данных, обработка данных и интерфейс для взаимодействия с пользователем. Обычно ПО, отвечающее за хранение данных, располагается на сервере. А пользовательский интерфейс – на стороне клиента. Обработка данных же распределяется между клиентом и сервером. Так, запрос пользователя на клиентской стороне необходимо обработать для передачи и дальнейшей обработки сервером, как и данные с сервера. При этом необходимо синхронизировать алгоритмы обработки данных на обоих сторонах, что значительно

усложняет процесс разработки. Ведь, как правило, разработка клиентской и серверной сторон ведется разными средствами (языками программирования и т.д.), а значит и разными специалистами.

Во избежание усложнения процесса разработки, бóльшая часть обработки данных выполняется на клиентской или серверной стороне.

В первом случае, при котором вычислительная нагрузка входит в обязанности клиентского ПО, система имеет следующие минусы:

– повышение нагрузки на сеть из-за передачи необработанных данных, которые могут содержать избыточную информацию, ненужную при представлении данных пользователю;

– усложнение поддержки и обновления всей системы, так как необходимо обновление каждого клиента, которых, как правило, намного больше, чем серверов;

– появление ошибок при обновлениях, если ПО обновляется не одновременно на всех клиентах;

– низкий уровень безопасности из-за того, что клиенты получают «лишнюю» информацию.

Во втором же случае, когда нагрузка ложится на плечи сервера, недостатки связаны с языками, используемыми при разработке серверных программ:

– СУБД-языки типа PL и SQL являются декларативными, то есть не имеют возможности пошаговой отладки, что значительно затрудняет разработку;

– производительность программ, написанных на СУБД-языках, значительно снижена, особенно при наличии в них сложных алгоритмов обработки данных;

Для решения вышеперечисленных проблем используются многоуровневые архитектуры клиент-сервер.

1.2. Многоуровневые архитектуры клиент-сервер

Многоуровневые архитектуры позволяют более оптимально распределить нагрузку. Например, нагрузка может распределяться между двумя серверами, один из которых выполняет задачи хранения данных, а другой – обработки. Серверы, обрабатывающие данные, принято называть серверами приложений, а серверы, хранящие данные, серверами баз данных. Пример многоуровневой системы клиент-сервер изображен на рис. 2.

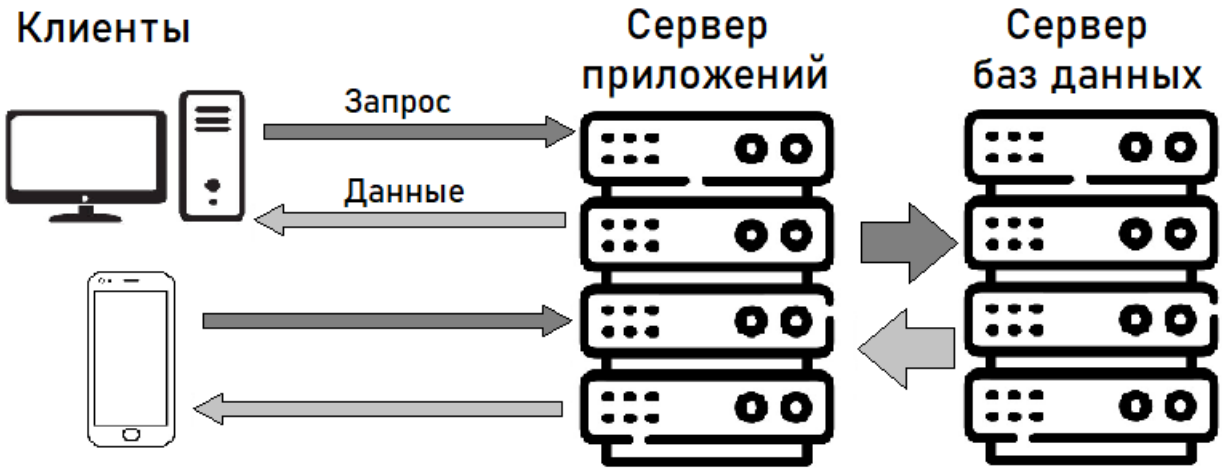


Рисунок 2 – Пример многоуровневой системы клиент-сервер

Таким образом, серверы приложений выполняют роль клиентов для серверов баз данных, а для фактического клиента – роль сервера. Кроме того, различные серверы приложений могут отвечать за разные функции обработки данных. А установление высокоскоростной связи между серверами не составляет труда, ведь чаще всего они находятся в одном здании.

Такие системы просто адаптируются к использованию в интернете. Фактический клиент имеет минимальное количество функций, связанных с обработкой данных, и может включать в себя лишь пользовательский интерфейс, который можно организовать с помощью универсального браузера, что заметно упрощает процесс разработки, обновления и унификации интерфейса для различных клиентов. Однако в таком случае необходимо дополнить сервер приложений Web-сервером, что также легко реализуемо, ведь

языки программирования, используемые для обработки и анализа данных, зачастую приспособлены для сетевого программирования (Java, Python и т.д.).

1.3. Серверы с установлением и без установления логического соединения

Кроме структуры разрабатываемой системы необходимо также выбрать один из двух типов взаимодействия клиент/сервер: с установлением или без установления логического соединения. Протоколы для реализации обоих типов входят в набор протоколов TCP/IP [3].

Взаимодействие с установлением логического соединения соответствует транспортному протоколу TCP. Данный протокол устраняет все проблемы, возникающие в процессе передачи, и обеспечивает полную надежность, выполняя следующие меры:

– проверка поступающих данных и автоматическая повторная передача неполученных сегментов;

– контрольное суммирование данных для проверки отсутствия их искажения во время передачи;

– последовательная нумерация пакетов для автоматического уничтожения их дубликатов и контроля их поступления в исходном порядке;

– управление потоком данных с целью исключения возможности передачи отправителем данных быстрее, чем их может принять получатель;

– информирование отправителя о том, что сеть передачи данных становится неработоспособной.

Структура пакета протокола TCP представлена на рис. 3.

Рисунок 3 – Структура пакета протокола TCP

Выполнение мер обеспечения надежности производится путем добавления в каждый пакет данных заголовка объемом 20 байт, который содержит следующие поля:

– порты источника и назначения;

– порядковый номер, который устанавливается приложением, инициирующим соединение (клиентом), увеличивается в соответствии с количеством переданных байтов;

– номер подтверждения, который принимающее устройство увеличивает в соответствии с количеством полученных байтов

– длина заголовка для того, чтобы система смогла понять, где начинаются данные;

– зарезервированное поле, которое всегда равно нулю;

– флаги для управления потоком данных в различных ситуациях, например, при сбросе текущей сессии связи;

– размер окна, в котором указывается объем данных, которые может принять отправитель;

– контрольная сумма, генерируемая отправителем для проверки ошибок принимающим приложением;

– срочный указатель для возможности пересылки некоторых байт вне очереди;

– поле опции используется расширения протокола или его тестирования.

Взаимодействие без установления логического соединения ассоциируется с транспортным протоколом UDP. Это протокол, наоборот, не предоставляет никаких гарантий надежной доставки данных. В вопросах надежности передачи UDP полагается на протокол более низкого уровня, то есть сетевой протокол IP. IP выполняет доставку пакетов, и его успешное функционирование зависит от работы базовых аппаратных и объединенных сетей и промежуточных шлюзов. Таким образом, протокол UDP обеспечивает бесперебойную передачу данных только в локальной сети, ведь в ней редко возникают ошибки. При использовании глобальной сети клиентское и серверное ПО, использующее транспортный протокол UDP, должно включать в себя код для обнаружения и исправления таких ошибок как: потеря пакетов, дублирование пакетов, задержка пакетов и доставка их в неверном порядке, отличающемся от исходного.

Резюмируя, можно сказать, что организация работы протокола TCP упрощает программирование и позволяет разработчику снять с себя ответственность за обнаружение и исправление ошибок при доставке данных. Протокол UDP в свою очередь же требует программного повышения надежности передачи, что зачастую представляет собой нетривиальную задачу для программиста, но при этом UDP обеспечивает бóльшую скорость передачи.

1.4. Серверы, поддерживающие и не поддерживающие состояние

Информация о состоянии – это обновляемые сервером данные о ходе взаимодействия с клиентом [4]. Серверы, не хранящие информацию о

состоянии, называются серверами, не поддерживающими состояние, а серверы, хранящие такую информацию – поддерживающими состояние.

Иногда наличие такой информации помогает уменьшить размеры сообщений, которыми обмениваются сервер и клиент, и, следовательно, увеличить, скорость отклика. Так, например, сервер может хранить информацию о предыдущих запросах каждого клиента, включающую в себя имя файла, с которым работал клиент. Это позволяет уменьшить размер последующих запросов клиента на запись в данном файле.

Отсутствие информации о состоянии может повысить надежность протокола, так как информация о состоянии может оказаться ошибочной, если сообщения будут потеряны, продублированы или доставлены в ошибочном порядке, или при аварийном перезапуске клиента. Таким образом ответа сервера клиенту может оказаться ошибочным.

1.5. Идентификация клиента

Еще одним важным для разрабатываемой системы клиент-сервер аспектом является вопрос идентификации клиента. В серверах, поддерживающих состояние, существуют два способа идентификации клиентов: оконечные точи и дескрипторы.

При применении идентификации по оконечным точкам сервер передает в ПО транспортного протокола требование предоставить информацию (например, IP адрес и номер порта протокола клиента) при поступлении запроса от клиента. Достоинство такой идентификации состоит в том, что она может выполняться автоматически, поскольку используемые при этом механизмы реализованы в транспортных протоколах, а не в прикладном.

Идентификация с помощью дескрипторов предполагает создание краткого идентификатора, который и называется дескриптором. При первичном запросе клиентом предоставляется полная информация о запросе, а сервер записывает полученную информацию и формирует для неё дескриптор, который обычно является простым целым числом. Затем сервер отправляет

дескриптор клиенту, который использует его в последующих запросах. Описанный метод не зависит от транспортного протокола, поэтому при сбросе соединения дескриптор остается действительным. Но для реализации такой идентификации необходимо задействовать само приложение-сервер и выделять память для хранения дескриптора.

Забегая вперед, стоит сказать, что в разработанной системе используется простейший вариант идентификации по оконечным точкам с помощью IP и номера порта, так как взаимодействие клиента и сервера происходит в идеальных условиях, то есть в локальной сети и при бесперебойной работе программ. Кроме того, при наличии небольшого количества клиентов, идентификация после восстановления связи не составляет труда. Однако в реальной объединенной сети, где программы могут аварийно перезагружаться, а сообщения могут быть потеряны, задержаны, продублированы и доставлены не в исходном порядке, проекты с сопровождением информации о состоянии требуют применения сложных прикладных протоколов и программных решений.

2. ИНТЕРФЕЙС СОКЕТОВ

2.1. Неформальная спецификация программного интерфейса протокола TCP/IP

Изначально протоколы TCP/IP были спроектированы с учетом применения их в самых различных средах (компьютерах, операционных системах, сетевых интерфейсных устройствах и т.д.). То есть в стандартах TCP/IP полностью исключены какие-либо спецификации интерфейса прикладного программирования в терминах средств ОС одного поставщика с целью исключения зависимости от внутреннего представления данных в этой ОС. Несмотря на это, интерфейс между TCP/IP и приложениями описан неформально в виде ряда выполняемых задач:

– распределение локальных ресурсов для обеспечения связи;
– задание локальной и удаленной оконечных точек связи;
– передача и прием данных;
– определение момента поступления данных;
– выработка и обработка срочных данных;
– завершение соединения;
– аварийное завершение связи и устранение последствий такого прекращения соединения;
– освобождение локальных ресурсов после завершения связи.

Также в интерфейсе должны быть реализованы операции лишь для клиента, такие как: инициирование соединения и передача дейтаграммы, или для сервера – ожидание входящего запроса на установление связи.

Роль интерфейса во взаимодействии приложений с встроенным в ОС ПО протоколов TCP/IP проиллюстрирована на рис. 4.

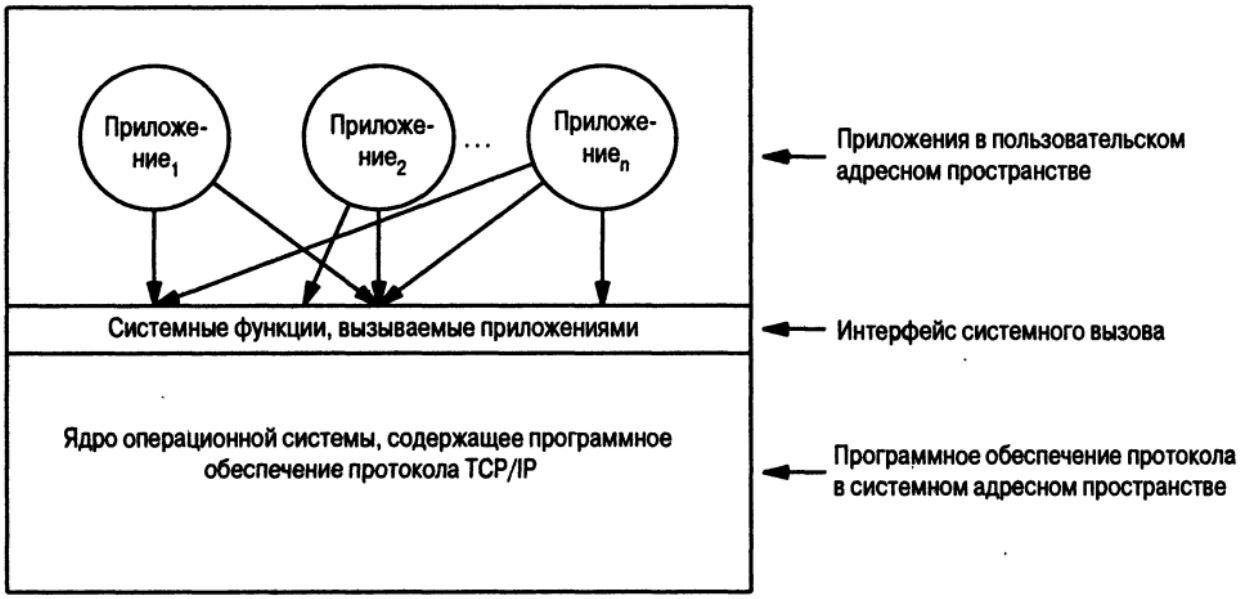


Рисунок 4 – Взаимодействие приложений с ПО протоколов TCP/IP через интерфейс системного вызова

При использовании в приложении системного вызова, который используется для передачи управления между приложением и процедурами ОС, управление передается интерфейсу системного вызова. Он, в свою очередь, передает управление ОС, которая направляет вызов внутренней процедуре, выполняющей нужную операцию. После выполнения операции управление через интерфейс возвращается к приложению.

При прохождении через интерфейс системного вызова процесс приобретает привилегии чтения и модификации структур данных в ОС, но сама ОС остается защищенной, так как необходимые операции выполняются с помощью процедур, написанных проектировщиками данной ОС.

2.2. Сокеты Беркли

Несмотря на то, что в стандартах TCP/IP полностью исключены какие-либо спецификации интерфейса взаимодействия с прикладными программами в зависимости от ОС, созданное в начале 1980-х годов ПО транспортных протоколов TCP/IP для операционной системы UNIX получило широкое распространение под названием интерфейса сокетов Беркли, так как разработка велась в Калифорнийском университете в городе Беркли.

Впоследствии интерфейс сокетов был включен во все операционные системы и стал использоваться на многих устройствах.

Сокет – это механизм, позволяющий создать интерфейс между прикладными программами и программным обеспечением протокола в операционной системе. При вызове соответствующей функции создания сокета, в памяти создается структура данных, приведенная на рис. 5.

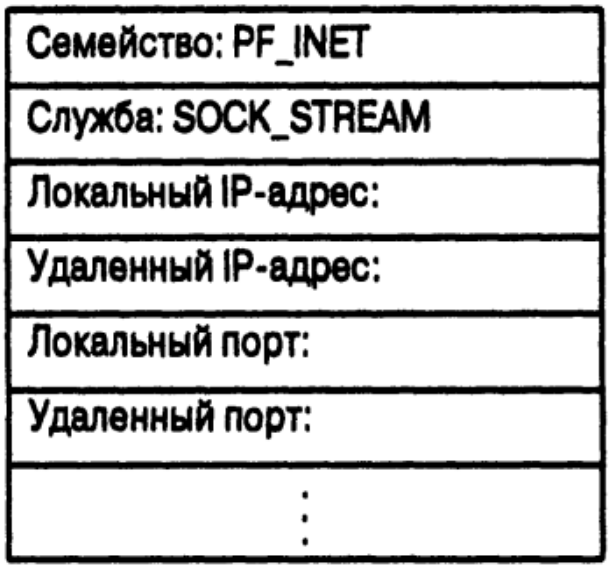
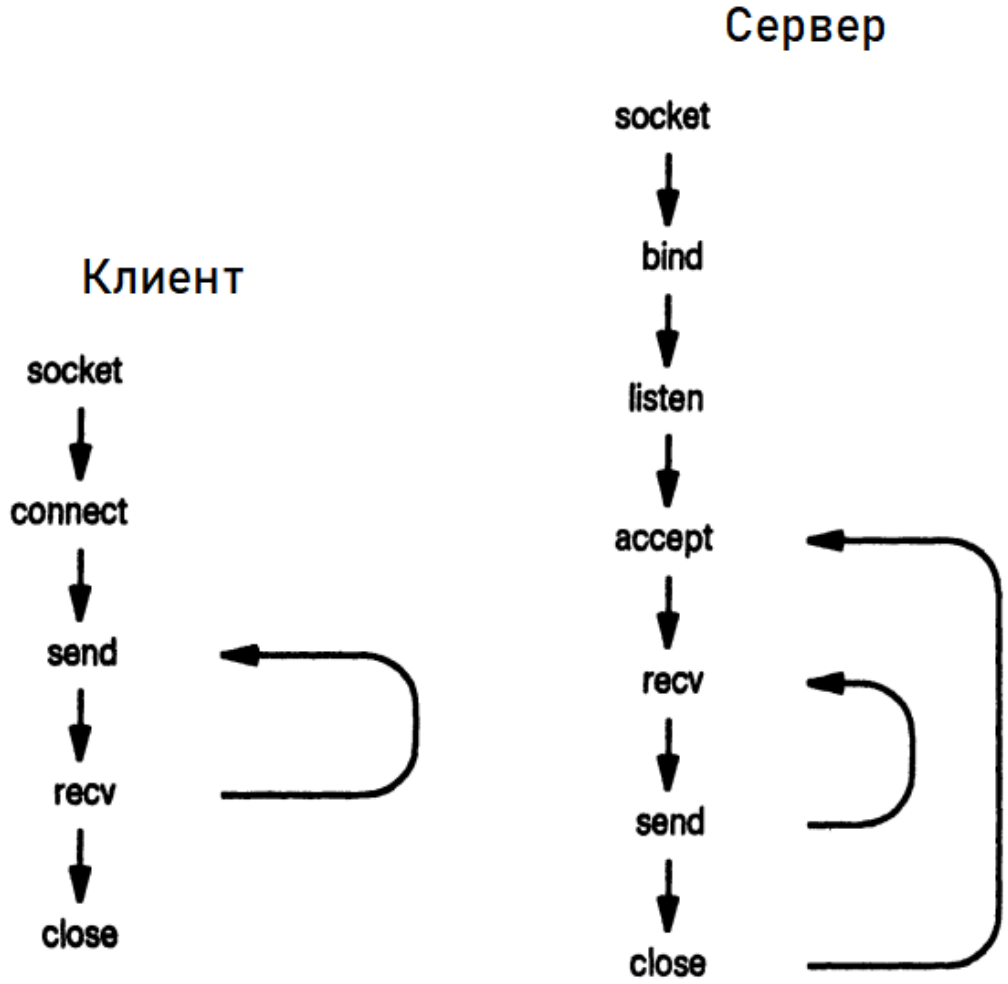


Рисунок 5 – Структура данных для сокета

При создании сокета указываются лишь семейство адресов, применяемых в TCP/IP (IPv4 или IPv6) и тип транспортного протокола (TCP или UDP). Заполнение прочих полей и взаимодействие с созданным сокетом происходит с помощью различных системных вызовов.

Так, например, рекомендованная последовательность вызовов для клиента и сервера представлена на рис. 6., *а* и на рис. 6., *б*, соответственно.



а *б*

Рисунок 6 – Пример последовательности системных вызовов сокетов для клиента (*а*) и для сервера (*б*)

Алгоритм действий сокета в серверной программе:

1) создание сокета с помощью функции socket;

2) привязка локальных IP адреса и порта к сокету с помощью функции bind;

3) установка сокета в пассивный режим, в котором он ждет входящие запросы вызовом функции listen;

4) вызов функции accept в цикле для перехода к ожиданию поступления очередного входящего запроса на установление соединения от клиента;

5) принятие следующего входящего байта данных (запросов от клиента) с помощью вызова функции recv в цикле, пока не закончится

принимаемая информация или не заполнится буфер, размер которого указывается при вызове функции

6) передача данных (ответов клиенту) путем вызова функции send;

7) разрыв соединения вызовов функции close и последующее ожидание поступления запроса на установление соединения.

Последовательность в клиентской программе:

1) создание сокета с помощью функции socket;

2) вызов функции connect для подключения к серверу с указанными IP и номером порта;

3) передача запросов с помощью функции send и получение ответов с помощью recv;

4) завершение связи вызовом функции close по окончании использования соединения.

Названия функций могут отличаться в различных ОС и языках программирования, однако всем им соответствуют одинаковые системные функции в ПО протокола TCP/IP в ядре ОС.

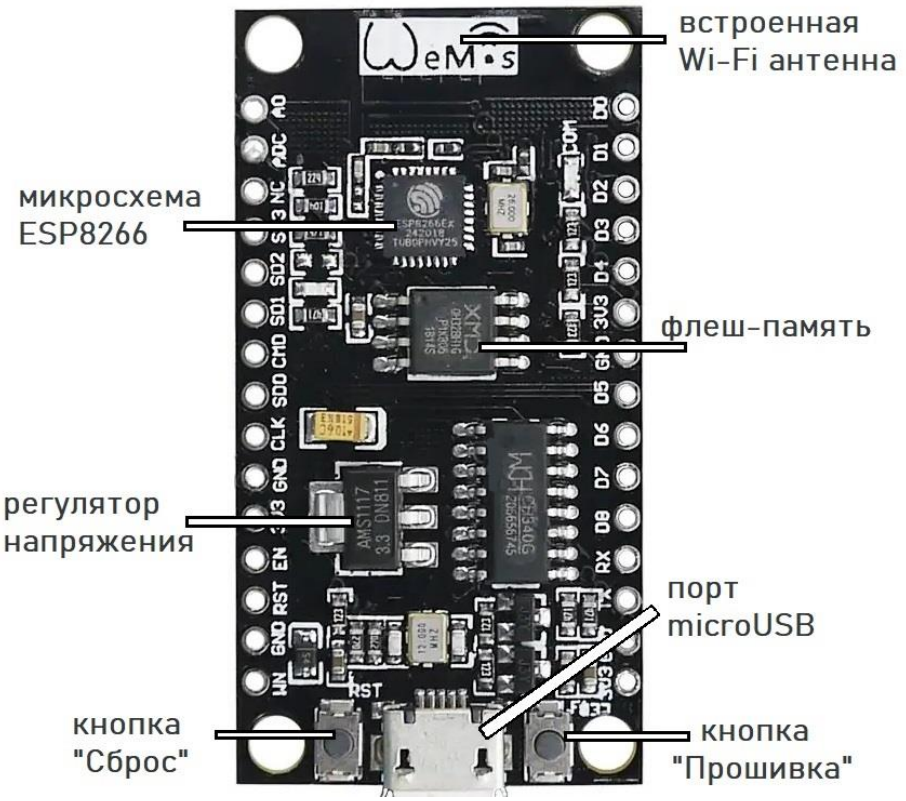
3. ОПИСАНИЕ УСТРОЙСТВ ЛАБОРАТОРНОГО МАКЕТА

3.1. Микроконтроллер ESP8266

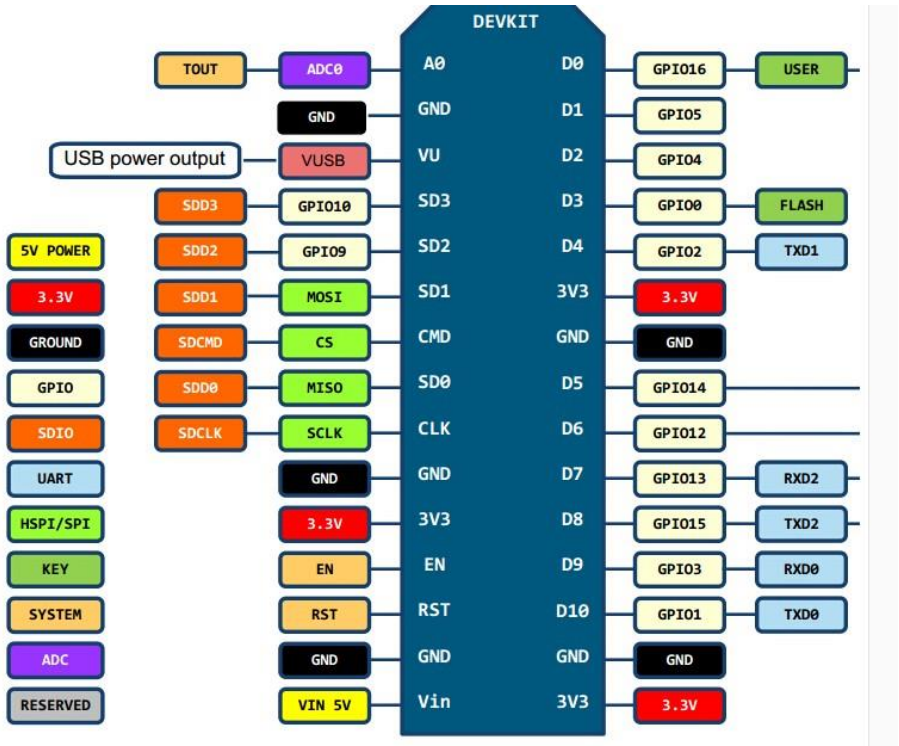
В качестве сервера в проектируемой системе сервер-клиент выступает МК ESP8266, установленный на плате проекта NodeMCU WeMos. Данная плата является самой распространенной для разработки различных устройств, использующих сетевое подключение Wi-Fi как в локальной сети, так и в глобальной сети интернет. На плате установлен уже упомянутый МК ESP8266, который работает на тактовой частоте 80 МГц, имеет 1 ядро и разрядность 32-бит [5]. Напряжение питания 5 В преобразуется в рабочее напряжение 3.3 В с помощью встроенного регулятора напряжения. Объем установленной флеш-памяти, на которую загружается исполняемая программа, – 4Мбайт. Разведенная на плате антенна Wi-Fi работает в диапазоне частот 2.4 ГГц – 2.5 ГГц и имеет выходную мощность 20 дБм, что обеспечивает связь в пределах нескольких десятков метров. Wi-Fi может работать в следующих режимах: клиент (ST), при котором плата подключается к существующей сети, точка доступа (AP) и клиент + программная точка доступа. Это позволяет построить на основе МК не только клиент-серверную архитектуру, а в том числе ячеистую, например при проектировании умного дома.

Плата имеет 2 аналоговых входа, подключенных к одному АЦП разрядностью 10 бит, что позволяет подключить к ней аналоговые датчики. Также на плате имеется 17 выводов общего назначения GPIO. Некоторые из них поддерживают интерфейс передачи данных I2C, который используется для взаимодействия контроллера с датчиками, что требуют высокую скорость и надежность обмена данными.

Кроме того, в качестве средства разработки для МК ESP8266 может использоваться Arduino IDE, позволяющее программировать ESP8266 так же легко как любые другие модули Arduino. Внешний вид платы WeMos NodeMCU Lua CH340 с указанием некоторых характерных модулей представлен на рис. 7, *а*. Обозначения выходов изображены на рис. 7, *б*.



а



б

Рисунок 7 – Плата WeMos NodeMCU Lua CH340: *а* – внешний вид; *б* – обозначения выходов

3.2. Источники и приемники информации на стороне сервера

В роли источников информации в составе стенда выступают датчики и сенсоры, подключенные к МК. В роли приемников выступают дискретные светодиоды и светодиодная лента, состоящая из восьми элементов. Далее приводится краткая характеристика каждого элемента макета.

Датчик BMP180 позволяет измерять атмосферное давление, температуру окружающей среды и высоту над уровнем моря [6]. В проектируемом стенде данный датчик используется для измерения давления. Диапазон измеряемых значений от 300 гПа до 1100 гПа, чего хватает для работы в домашних условиях. Напряжение питания от 3.3 до 5 В, которое может обеспечить используемая плата.

Модуль GY-291 – трехосевой акселерометр на базе чипа ADXL345. Датчик способен измерять статическое ускорение по трём осям, вызванное гравитацией. Также требует напряжение питания от 3.3 до 5 В [7].

Датчик APDS9960 способен определять уровень освещенности по трём составляющим спектра (красный, синий и зеленый), общий уровень освещенности, приближение и отдаление объектов и жесты (движения относительно датчика). С помощью встроенных четырёх светодиодов реагирует на следующие жесты:

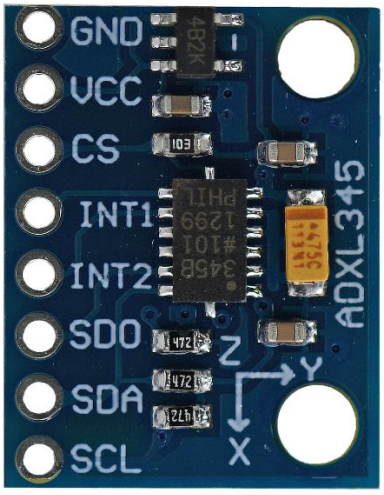
- UP - Движение руки перед датчиком вперёд или вверх в зависимости от положения датчика в пространстве.
- DOWN - Движение руки перед датчиком назад или вниз в зависимости от положения датчика в пространстве.
- LEFT - Движение руки перед датчиком влево.
- RIGHT - Движение руки перед датчиком вправо.
- NEAR - Приближение руки к датчику.
- FAR удаление руки от датчика.

В отличие от прочих используемых датчиков требует питание только 3.3 В, а также отдельное дополнительное питание ИК светодиода (VL) от 3

до 4.5 В. В разрабатываемом макете данный датчик выполняет функцию распознавателя жестов.

Все перечисленные датчики имеют I2C интерфейс, с помощью которого их можно подключить к МК. Шина I2C синхронная и состоит из двух двунаправленных линий связи: линия данных (SDA) и линия тактирования (SCL). Для подключения этих линий используются входы D2 и D1, соответственно. Внешний вид датчиков изображен на рис. 2. Структурная схема стенда представлена на рис. 3.

а



б

в

Рисунок 8 – Используемые датчики: *а* – барометр BMP180; *б* – акселерометр ADXL345; *в* – датчик жестов APDS9960

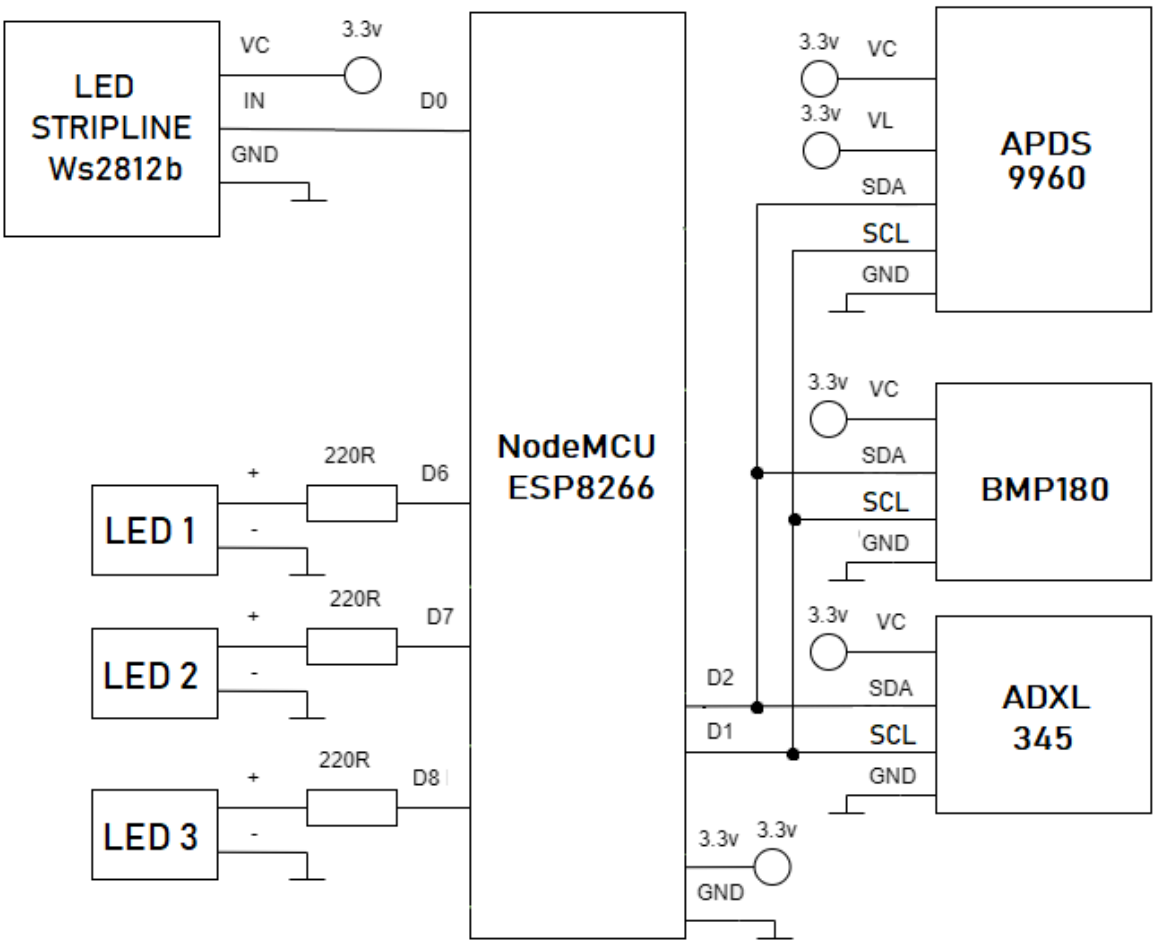


Рисунок 9 – Структурная схема проектируемого стенда

Кроме рассмотренных датчиков в состав стенда входят дискретные светодиоды, имеющие два состояния: логический ноль или единица, и светодиодная лента, каждый из восьми элементов которой может принимать разные цвета. Светодиоды выполняют роль приемников информации и подключены к цифровым выходам D6, D7, D8 (дискретные светодиоды) и D0 (светодиодная лента).

Итак, рассмотрев макет сервера в проектируемой структуре, можно приступить к разработке программы для управления стендом.

4. РАЗРАБОТКА СЕРВЕРНОЙ ПРОГРАММЫ ДЛЯ УПРАВЛЕНИЯ МК

4.1. Функции программы-сервера

Классифицируя разрабатываемую систему на основе подразделов 1.1 и 1.2 данной работы, её можно отнести скорее к многоуровневой архитектуре клиент-сервер, функции сервера приложений и сервера БД, в которой выполняет МК. То есть, в реализуемой системе МК-ПК вычислительная нагрузка распределена равномерно между сервером и клиентом

Так, на сервер, в роли которого выступает стенд под управлением МК, кроме сетевого взаимодействия неминуемо ложится обязанность по преобразованию получаемых с датчиков значений в удобный для беспроводной передачи вид, чтобы минимизировать нагрузку на сеть, возникновение ошибок и упростить последующий парсинг информации.

В качестве среды разработки для ESP8266 была выбрана Arduino IDE, так как она широко используется для программирования любых Arduino плат, имеет большое количество подключаемых библиотек, в том числе для управления устройствами в разрабатываемом стенде, для работы с сетями Wi-Fi, пакетами JSON. Вдобавок, язык программирования, используемый в ней для написания кода, основан на языке C++, который широко распространен и является одним из самых популярных языков программирования в мире [9].

Схема взаимодействия стенда под управлением МК и ПК по локальной беспроводной сети Wi-Fi представлена на рис. 10.

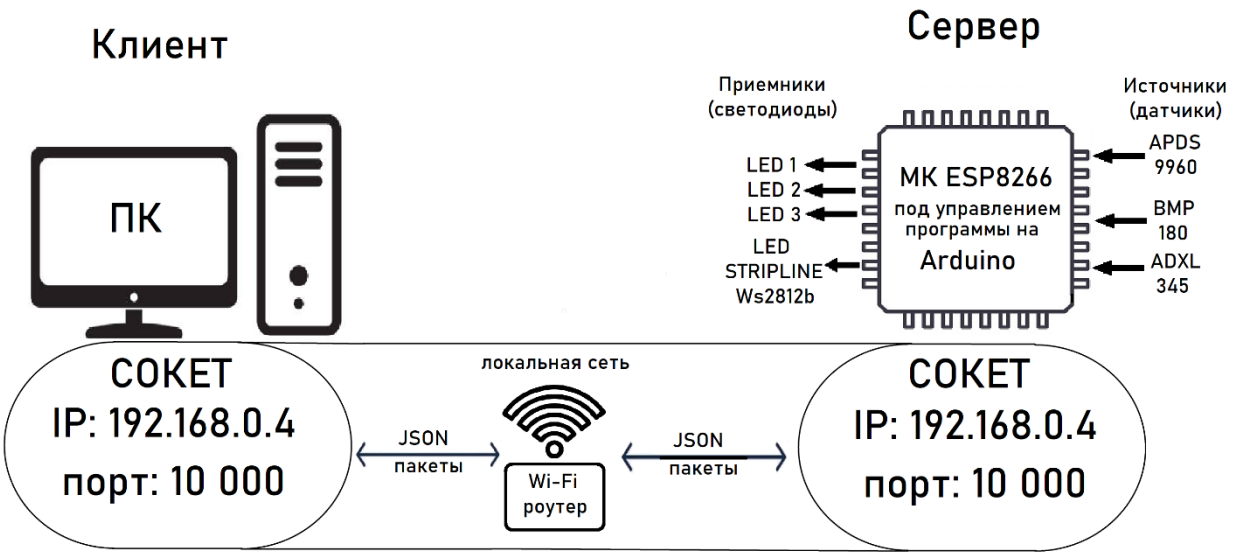


Рисунок 10 – Схема взаимодействия МК и ПК

Опираясь на первые разделы данной работы, можно сформулировать функции, выполняемые МК под управлением разрабатываемой программы:

1) создание сервера, способного выполнять прием и передачу данных по локальной сети Wi-Fi, и подключение его к сокету с указанным портом;

2) прием и обработка поступающих с ПК по TCP/IP протоколу JSON пакетов, включающих в себя сигналы для управления дискретными светодиодами и светодиодной лентой;

3) управление светодиодами согласно принятым и обработанным данным с ПК;

4) обработка показаний подключенных датчиков давления, ускорения и жестов для последующей передачи их по беспроводной сети на ПК.

4.2. Организация сетевого взаимодействия

Взаимодействие между серверной программой и транспортным протоколом TCP/IP происходит с помощью сокета, подключенного к указанному порту.

Для реализации сетевого взаимодействия была использована библиотека «ESP8266WiFi.h», позволяющая МК подключаться к точке доступа Wi-

Fi, создавать серверы и сокеты, подключенные к выбранным портам, побайтно принимать и передавать информацию и т.д. [10].

Для создания сервера с сокетом необходимо создать объект класса «WiFiServer», в качестве аргумента передав порт, к которому он будет подключен. Подключение к точке доступа Wi-Fi производится с помощью вызова метода «WiFi.begin», в качестве аргументов которому передаются SSID и пароль точки доступа. Также для начала работы сервера необходимо вызвать у него метод begin. Создание сервера и подключение МК к точке доступа представлено на рис. 11.

```
39 #include <ESP8266WiFi.h>    // библиотека для создания и работы с Wi-Fi сервером
40
41 const char* ssid = "RT-GPON-B750";   // SSID
42 const char* password = "HFmk4UdK";  // пароль
43
44 WiFiServer server(10000); // создание сервера, что общается по указанному порту
45
46 void setup()
47 {
48   WiFi.begin(ssid, password);    // поключение к точке доступа
49   server.begin();   // сервер начинает "слушать" входящие запросы
```

Рисунок 11 – Подключение МК к точке доступа и создание сервера

Стоит упомянуть, что вызов указанных команд для начала работы Wi-Fi сервера и подключения его к сети находится в функции setup, которая выполняется лишь единожды после включения МК. К тому же, подключение к точке доступа, первоначальная настройка датчиков и светодиодов, которая будет описана ниже, выполняются в функции setup при запуске макета.

Для упрощения пользовательского интерфейса на ПК и уменьшения нагрузки на сеть, проверка, диагностика датчиков и получение IP в локальной сети производятся при подключении МК к ПК через microUSB и общению через интерфейс «Serial». В качестве примера на рис. 12 и рис. 13, соответственно, приведены код программы для ESP8266 и алгоритм, который выполняет первоначальное подключение и получение IP адреса в локальной сети.

```
16 void setup()
17 {
18   Serial.begin(115200);
19
20   WiFi.begin(ssid, password);     // поключение к точке доступа
21
22   while (--30 && WiFi.status() != WL_CONNECTED)
23   {
24     delay(500);
25     Serial.println(".");
26   }
27
28   if (WiFi.status() != WL_CONNECTED)
29   {
30     Serial.println("Не удалось подключиться к сети, перезапустите устройство.");
31   }
32   else
33   {
34     Serial.println("");
35     Serial.println("Подключение к сети WiFi прошло успешно");
36     Serial.println("IP адрес в локальной сети: ");
37     Serial.println(WiFi.localIP());    // IP адрес платы в локальной сети роутера
38   }
39 }
```

Рисунок 12 – Первоначальное подключение МК к точке доступа Wi-Fi

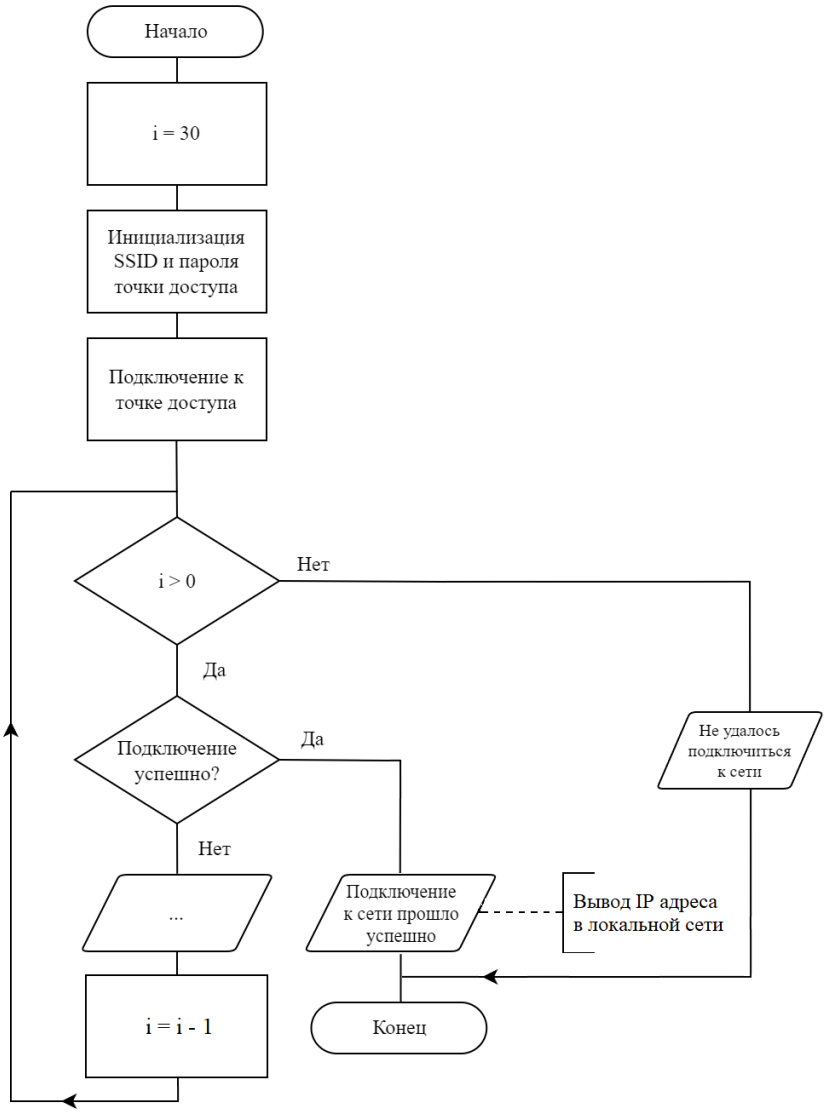


Рисунок 13 – Алгоритм первоначального подключения МК к точке доступа Wi-Fi

4.3. Прием данных с ПК

Принимаемыми микроконтроллером данными являются сигналы управления дискретными светодиодами и светодиодной лентой. Для управления светодиодами объявляются переменные для обращения к ним с указанием выходов на плате NodeMCU Lua CH340, к которым они подключены. Для управления светодиодной лентой используется подключаемая библиотека «FastLED.h». В коде программы создается объект класса CRGB, входящего в ранее подключенную библиотеку, и массив принятых кодов цветов для взаимодействия с лентой, а также указывается количество светодиодов на ней и используемый выход. Создание объектов для управления приемными устройствами изображено на рис. 14.

```
22 #include <FastLED.h>    // библиотека для управления светодиодной лентой
23
24 //объявление констант (названий выходам 12 (D6), 13 (D7), 15 (D8))
25 #define PIN_LED1 12
26 #define PIN_LED2 13
27 #define PIN_LED3 15
28
29 #define NUM_LEDS_IN_STRIPLINE 8   // количество светодиодов на ленте
30 #define DATA_PIN 16   // пин для подключения ленты
31 CRGB ledsLine[NUM_LEDS_IN_STRIPLINE];   // экземпляр класса CRGB
32 int stripColors[NUM_LEDS_IN_STRIPLINE]; // массив цветов светодиодов в ленте
```

Рисунок 14 – Создание объектов для управления светодиодами

В функции «setup» выполняется перевод цифровых выходов, к которым подключены дискретные светодиоды, в режим вывода и настройка светодиодной ленты, что представлено на рис. 15. Настройка ленты включает в себя:

– указание используемой модели светодиодной ленты;
– указание выхода платы, к которому подключена лента;
– указание созданного ранее объекта ленты в программе;
– обозначение количества светодиодов в ленте;
– задание цветокоррекции и яркости.

```
48 // перевод цифровых портов со светодиодами в режим вывода
49    pinMode(PIN_LED1, OUTPUT);
50    pinMode(PIN_LED2, OUTPUT);
51    pinMode(PIN_LED3, OUTPUT);
52 // настройка светодиодной ленты
53    FastLED.addLeds <WS2812, DATA_PIN, GRB>(ledsLine, NUM_LEDS_IN_STRIPLINE).setCorrection(TypicalLEDStrip);
54    FastLED.setBrightness(50);
```

Рисунок 15 – Настройка приемных устройств

Перед началом приема данных необходимо создать объект типа «WiFiClient», который будет соответствовать подключенному клиенту. Стоит отметить, что полученная информация о соединении сохраняется в памяти, даже когда связь с клиентом прерывается. То есть при восстановлении связи с клиентом, соединение автоматически восстановится. Кроме того, все приведенные ниже команды находятся в функции «loop», которая выполняется непрерывно после «setup», что значит, что соединение и принимаемые данные постоянно обновляются. Последовательность вызовов функций сокета во всей серверной программе составлена в соответствии с рекомендациями, приведенными в подразделе 2.2. (см. рис. 6, б).

Сперва проводится проверка, соединен ли в данный момент клиент с помощью вызова у него метода «connected». Стоит отметить, что клиент считается подключенным, даже если соединение было разорвано, но все еще есть непрочитанные данные. Таким образом уменьшается риск потери получаемых данных.

Перед непосредственным принятием и записью полученной информации производится проверка сокета на наличие принятых байтов с помощью вызова метода «available» у клиента.

Наконец, побайтовое считывание и запись получаемой от клиента информации в заранее созданный массив выполняется вызовом метода read у клиента.

Фрагмент кода программы, выполняющий прием данных, представлен на рис. 16., а алгоритм – на рис. 17.

```
59   // получение и запись клиента
60   WiFiClient client = server.available();
61 // проверка, существует ли клиент
62   if (client)
63   {
64     while (client.connected())  // проверка, подключен ли клиент
65     {
66       while (client.available() > 0)  // проверка, есть ли байты для чтения от клиента
67       {
68         data[i] = static_cast<char>(client.read()); // считываем и записываем первый на очереди байт
69         i++;
70       }
```

Рисунок 16 – Прием данных с ПК

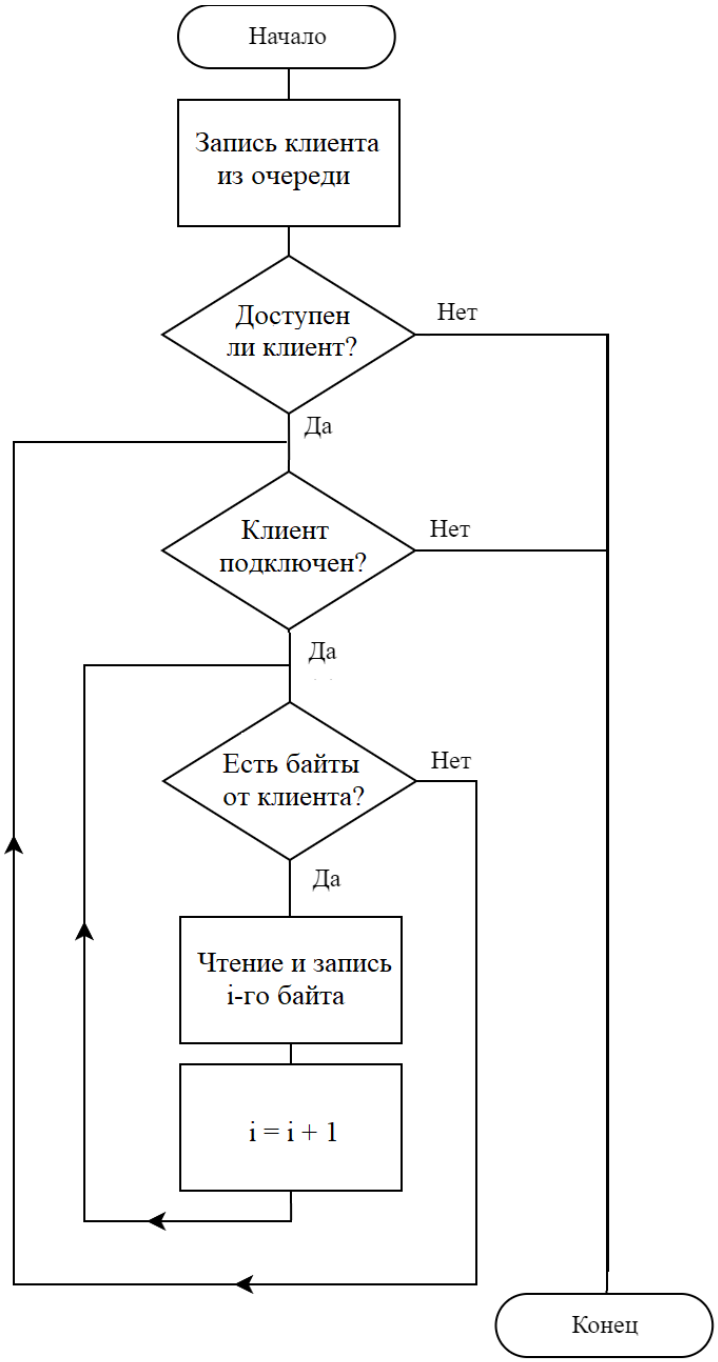


Рисунок 17 – Алгоритм приема данных с ПК

4.4. Обработка данных с ПК

Входящими данными являются сигналы для дискретных светодиодов (логический ноль или единица) и массив кодов цветов для светодиодной ленты (1 – красный, 2 – синий, 3 – зеленый). Для хранения и передачи таких данных отлично подходит текстовый формат обмена данными JSON. JSON основан на парах ключ/значение, которые образуют упорядоченный список значений [11]. При этом ключ – это название передаваемого значения, а само значение может быть числом, строкой, булевой переменной, объектом или массивом. Пример принимаемого микроконтроллером JSON пакета в разрабатываемой системе представлен на рис. 18.

```
{
  "LED1": 1,
  "LED2": 0,
  "LED3": 1,
  "LEDSTRIP": [1, 2, 3, 1, 2, 3, 1, 2]
}
```

Рисунок 18 – Принимаемый МК JSON пакет

Процесс преобразования строки в объект JSON называется сериализацией, а обратный процесс – десериализацией. Для работы с объектами JSON была использована библиотека «ArduinoJson.h».

Десериализация принятого JSON пакета начинается после заполнения массива с фиксированной длиной, так как байтовый размер принимаемого пакета всегда постоянен и известен (70 байт). В результате десериализации получается структура, из которой можно извлечь значения, которые соответствуют конкретным ключам. Этот процесс называется парсингом. Извлеченные данные передаются в переменные, которые в последствии используются в качестве аргументов при подаче сигналов на цифровые выходы, к которым подключены дискретные светодиоды, с помощью команды «digitalWrite». Также извлекается массив кодов цветов для светодиодной ленты, и с помощью созданной функции (см. рис. 19) им сопоставляются фактические цвета, подаваемые на ленту светодиодов. Фрагмент кода программы, ответственный за десериализацию и парсинг входящего JSON пакета, изображен на рис. 18.

```
66      DynamicJsonDocument parsed(1024); // выделение памяти для принятой JSON строки
67      deserializeJson(parsed, data);  // десериализация принятого JSON пакета
68      //парсинг пакета JSON
69      // получение сигналов на дискретные светодиоды
70      int LED1_status = parsed["LED1"];
71      int LED2_status = parsed["LED2"];
72      int LED3_status = parsed["LED3"];
73      // получение массива сигналов на светодиодную ленту
74      for (int i = 0; i < NUM_LEDS_IN_STRIPLINE - 1; i++)
75      {
76        stripColors[i] = parsed["LEDSTRIP"][i];  // запись цветов светодиодов в ленте
77      }
78      // смена цветов дискретных светодиодов
79      digitalWrite(PIN_LED1, LED1_status);
80      digitalWrite(PIN_LED2, LED2_status);
81      digitalWrite(PIN_LED3, LED3_status);
82      // смена цветов светодиодов в ленте
83      for (int a = 0; a < NUM_LEDS_IN_STRIPLINE; a++)
84      {
85        changeLedInStrip(a, stripColors[a]);
86      }
```

Рисунок 18 – Обработка полученного МК пакета JSON

```
146 // функция смены цветов светодиодной ленты
147 void changeLedInStrip(int ledPos, int ledColor)
148 {
149   switch (ledColor)
150   {
151     case 1:
152       ledsLine[ledPos] = CRGB::Red;
153       FastLED.show();
154       break;
155     case 2:
156       ledsLine[ledPos] = CRGB::Green;
157       FastLED.show();
158       break;
159     case 3:
160       ledsLine[ledPos] = CRGB::Blue;
161       FastLED.show();
162       break;
163   }
164 }
```

Рисунок 19 – Функция смены цветов светодиодов на ленте

4.5. Обработка и передача показаний датчиков

В качестве передаваемых клиенту с сервера данных выступают показания следующих датчиков: барометр BMP180, акселерометр ADXL345 и датчик жестов APDS9960. Для работы с ними использовались библиотеки «SFE_BMP180.h», «Adafruit_ADXL345_U.h» «SparkFun_APDS9960.h»,

соответственно. Кроме того, были подключены такие библиотеки как: «Wire.h» для работы с интерфейсом I2C и «Adafruit_Sensor.h» для управления библиотеками производителя ПО Adafruit.

Для управления датчиками сперва необходимо создать соответствующие им экземпляры классов из подключенных библиотек, что представлено на рис. 19.

```
10 // создание экземпляра класса SFE_BMP180 для работы с барометром
11 SFE_BMP180 pressure;
12 // создание объекта, экземпляр класса ADXL345_U для работы с акселерометром
13 Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);
14 // создание объекта, экземпляр класса SparkFun_APDS9960 для работы с датчиком жестов
15 SparkFun_APDS9960 apds = SparkFun_APDS9960();
```

Рисунок 20 – Создание объектов для управления датчиками

Для записи в JSON-объект конструкций ключ/значение, необходимо сначала создать данный объект, указав в качестве аргумента количество выделяемых байт. Получаемый в результате пакет может иметь разный объем, так как показания датчиков давления и ускорения – это числа с плавающей запятой, а показания датчика жестов – это строка с названием текущего жеста, которая может иметь разный объем (DIR_LEFT, DIR_UP, DIR_FAR и т.д.). Поэтому необходимо выделить память для максимально большого возможного объема получающейся структуры JSON.

Получение показаний барометра выполняется с помощью вызова функции «getPressure», которая представлена на рис. 21.

```
119 // функция измерения давления
120 double getPressure()
121 {
122   char status;
123   double T, P;
124   pressure.startTemperature(); // запуск измерения температуры
125   delay(status);  // ожидание завершения измерений
126   pressure.getTemperature(T);  // извлечение данных о температуре
127   pressure.startPressure(3);  // запуск измерения давления
128   delay(status);
129   pressure.getPressure(P, T); // извлечение данных о давлении
130   return (P);   //возвращение измеренного значения в мбар (гПа)
131 }
```

Рисунок 21 – Измерение давления с учетом температуры

Показания акселерометра считываются через обращение к методам объекта, хранящего эти показания. А для чтения текущего жеста вызывается функция «readGesture».

После получения показаний всех датчиков в созданном JSON объекте создаются пары ключ/значение, где в качестве ключей выступают названия измеряемых величин, а в качестве значений – результаты измерения. Пример отправляемого микроконтроллером пакета изображен на рис. 22.

```
{
  "Pressure": 1001.65,
  "Acceleration": [-0.86, 0.04, 8.32],
  "Gesture": "DIR_LEFT"
}
```

Рисунок 22 – Отправляемый на ПК пакет JSON

Затем, отправляемые данные сериализуются в JSON структуру, которая уже имеет конкретный объем в байтах и отправляется по одному байту клиенту с помощью вызова метода «write». Кроме созданного пакета после отправки всех байт также выполняется отправка пустого блока, чтобы клиентское приложение на ПК смогло различить прибывающие непрерывно пакеты. Фрагмент кода программы, обрабатывающий и передающий по сети показания датчиков представлен на рис. 23.

```
//  выделение памяти для отправляемого JSON объекта
      DynamicJsonDocument doc(64);

      double Pr = getPressure();  // измерение давления
// создание структуры для занесения в неё показаний акселерометра
      sensors_event_t event;
// получение текущих показаний акселерометра
      accel.getEvent(&event);
      double aX = event.acceleration.x;
      double aY = event.acceleration.y;
      double aZ = event.acceleration.z;

// получение текущих показаний датчика жестов
      gesture = apds.readGestur();

// создание пар ключ-значение (сенсор-показания) в структуре JSON
      doc["Pressure"] = Pr; // давление в гПа
      doc["Acceleration"][0] = aX;  // ускорение X в м/с^2
      doc["Acceleration"][1] = aY;  // ускорение Y в м/с^2
      doc["Acceleration"][0] = aZ;  // ускорение Z в м/с^2
      doc["Gesture"][0] = gesture;  // жест

      String output;  // создание переменной для хранения JSON пакета
      serializeJson(doc, output);   // запись JSON пакета в ранее созданную переменную
      for (int i = 0; i < output.length(); i++)
      {
        client.write(output[i]);
      }
      client.write("   ");
```

Рисунок 23 – Обработка и передача показаний датчиков

Также стоит упомянуть, что обработка и передача данных выполняется сразу после приема и обработки пакетов с ПК в функции «loop», то есть показания датчиков обновляются непрерывно.

Полный код программы управления МК приведен в приложении А, а алгоритм программы – в приложении Б.

5. РАЗРАБОТКА КЛИЕНТСКОЙ ПРОГРАММЫ ДЛЯ ПК

5.1. Функции программы-клиента

Клиентское приложение во многом схоже с серверной программой. Главное отличие состоит в том, что клиент является инициатором установления соединения клиент-сервер, для чего ему необходимо знать номер порта сокета сервера и IP, который присвоен серверу на МК в локальной сети. Вдобавок клиентское приложение должно обрабатывать принимаемые данные для последующего их вывода в пользовательский графический интерфейс (GUI).

В качестве языка программирования для разработки приложения-клиента для ПК под управлением Windows был выбран Python, так как с его помощью можно легко создать простейший GUI. Вместе с тем он имеет большое количество подключаемых библиотек, который позволяют работать с сокетами, беспроводными сетями и JSON пакетами.

Дополненная с учетом задач клиента схема взаимодействия МК и ПК в клиент-серверной архитектуре представлена на рис. 24.

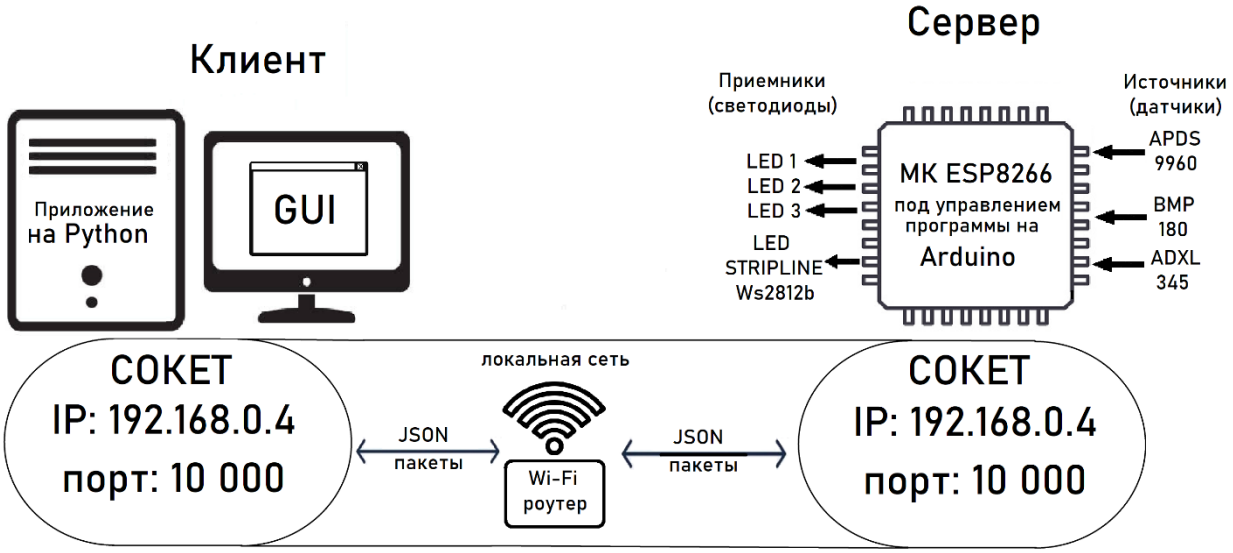


Рисунок 24 – Схема сетевого взаимодействия ПК и МК

5.2. Создание графического пользовательского интерфейса

Создаваемая программа с GUI является событийно ориентированной. То есть та или иная часть программного кода начинает выполняться лишь тогда, когда случается какое-либо событие, например, нажатие кнопки. Поэтому разработку приложения стоит начать с создания GUI.

Для создания графического пользовательского интерфейса был использован пакет «Tkinter», предназначенный для работы с библиотекой Tk, которая содержит компоненты GUI [13]. Под компонентами GUI подразумеваются окна, кнопки, текстовые поля для ввода/вывода, скроллеры, списки, радиокнопки, флажки и другие элементы, через которые происходит взаимодействие пользователя с программой. Все эти элементы графического интерфейса называются виджетами.

Создание виджета происходит путем обращения к конструкции с его названием («Label», «Checkbutton» и др.) в модуле «ttk», импортированного из пакета tkinter. В качестве аргументов функции указываются различные параметры создаваемых виджетов, такие как текст, размеры и привязанная переменная, хранящая текущее значение, выводимое виджетом. Фрагмент кода, соответствующий созданию необходимых для программы виджетов представлен на рис. 25.

```
# создание виджетов типа Entry для вывода показаний датчиков
valP = ttk.Entry(textvariable=p)
valAx = ttk.Entry(textvariable=aX)
valAy = ttk.Entry(textvariable=aY)
valAz = ttk.Entry(textvariable=aZ)
valGest = ttk.Entry(textvariable=gest)

# создание виджетов типа Label для текстовых меток
labelRecieved = ttk.Label(text="Принимаемые данные")
labelP = ttk.Label(text="Давление, гПа")
labelAx = ttk.Label(text="Ускорение X, м/с^2")
labelAy = ttk.Label(text="Ускорение Y, м/с^2")
labelAz = ttk.Label(text="Ускорение Z, м/с^2")
labelG= ttk.Label(text="Жест")
labelSend = ttk.Label(text="Отправляемые данные")
labelLed1 = ttk.Label(text="Светодиод №1")
labelLed2 = ttk.Label(text="Светодиод №2")
labelLed3 = ttk.Label(text="Светодиод №3")
labelLedS = ttk.Label(text="Выберите цвета для светодиодной ленты")

# создание виджетов типа Checkbutton для управления светодиодами
btnL1 = ttk.Checkbutton(text="вкл/выкл", variable=statLed1)
btnL2 = ttk.Checkbutton(text="вкл/выкл", variable=statLed2)
btnL3 = ttk.Checkbutton(text="вкл/выкл", variable=statLed3)

# создание виджетов типа Combobox для управления светодиодной лентой
comboLS1 = ttk.Combobox(values=ledStripColors, width=8, state="readonly")
comboLS2= ttk.Combobox(values=ledStripColors, width=8, state="readonly")
```

Рисунок 25 – Создание виджетов в окне приложения

Размещение виджета в окне приложения осуществляется с помощью вызова метода «place», принимающего в качестве параметров смещение элемента по горизонтали и вертикали в пикселях, соответственно, относительно верхнего левого угла контейнера. Пример размещения виджетов в окне показан на рис. 26.

```
# размещение виджетов в окне
labelRecieved.place(x=50, y=10)
labelP.place(x=10, y=50)
labelP.place(x=10, y=50)
labelAx.place(x=10, y=100)
labelAy.place(x=10, y=150)
labelAz.place(x=10, y=200)
labelG.place(x=10, y=250)
```

Рисунок 26 – Пример размещения созданных виджетов в окне приложения

Кроме выполнения вышеупомянутых команд, необходимо также создать объект окна перед размещением в нем виджетов с помощью конструктора «Tk» и запустить его после указания всех функций, вызвав у объекта метод «mainloop». Созданное окно приложения представлено на рис. 27.

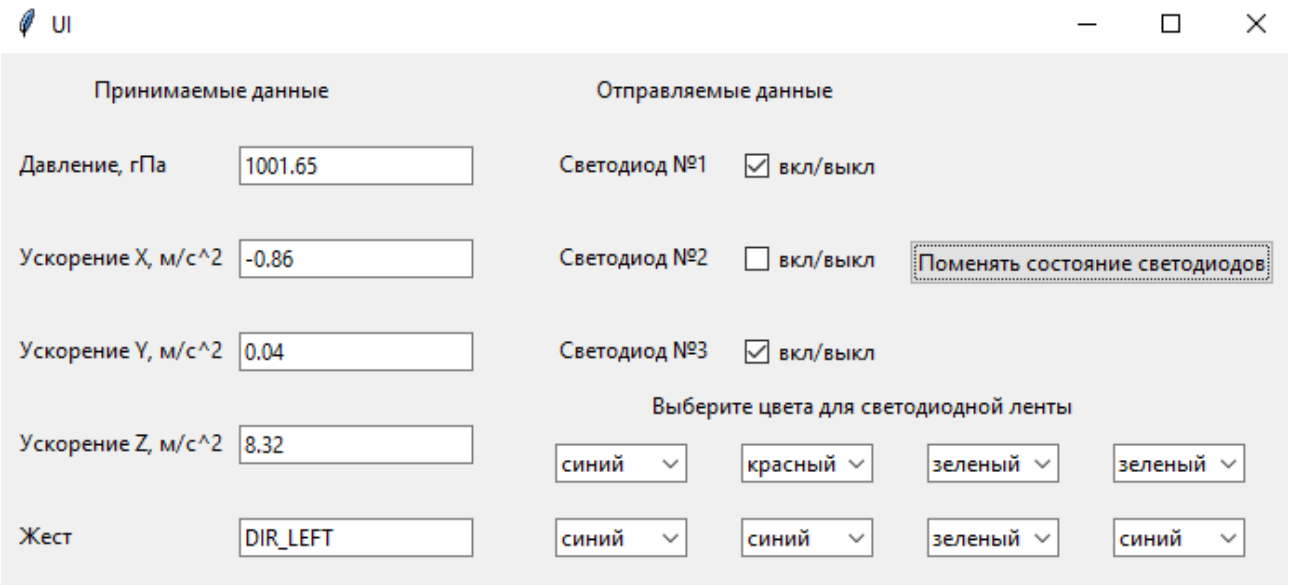


Рисунок 27 – Окно приложения-клиента

5.3. Прием данных с МК

Как уже было упомянуто в разделе 2 и при описании разработки программы для МК механизмом создания интерфейса между прикладной программой и транспортным протоколом TCP/IP является сокет. Для реализации сокетов была использована встроенная в Python библиотека «socket», позволяющая создавать сокеты, привязывать их к IP адресам и портам, читать и отправлять данные через сокет и т.д. [14].

Для начала необходимо создать сокет с помощью функции «socket», которая в качестве аргументов принимает:

– используемое семейство адресов: «AF_UNIX» для ОС Linux, «AF_INET» для IPv4 и «AF_INET» для IPv6;

– транспортный протокол: «SOCK_STREAM» для TCP и «SOCK_DGRAM» для UDP.

В результате подключения МК к источнику Wi-Fi, ему присваивается адрес IP в локальной сети в формате IPv4, поэтому было выбрано данное семейство адресов при объявлении сокета. Протокол TCP обеспечивает надежность передачи данных, предотвращая потери пакетов, их дублирование, задержку и прибытие в неверном порядке. В разрабатываемом макете точность

получаемых данных важнее, чем скорость их передачи, которую может обеспечить транспортный протокол UDP, так что было принято решение использовать TCP протокол для установки логического соединения между клиентом и сервером.

Следующим шагом является подключение созданного сокета к IP адресу сервера на микроконтроллере и нужному сокету. Подключение производится вызовом метода «connect». Создание и подключение сокета представлено на рис. 28.

```
6   ip = '192.168.0.4'  # IP адрес платы в локальной сети роутера
7   port = 10000  # выбранный для общения порт
8
9   conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # создание сокета
10  conn.connect((ip, port))  # соединение с сокетом с заданным адресом и портом
```

Рисунок 28 – Создание и подключение сокета на ПК

Для приема поступающих на сокет данных, у него вызывается метод «recv», аргументом для которого является количество принимаемых байт. При вызове этого метода выполнение программы останавливается, пока не прекратится поток пакетов данных, что недопустимо для создаваемого приложения, ведь оно должно выполнять и другие задачи кроме приема данных, которые поступают непрерывно. Поэтому перед приемом данных необходимо переключить сокет в неблокирующий режим работы, вызвав у него метод с указанием аргумента «setblocking(False)». Функция, принимающая и обрабатывающая входящие данные изображена на рис. 29.

Последовательность вызовов функций сокета во всей клиентской программе выбрана в соответствии с рекомендациями, приведенными в подразделе 2.2 (см. рис. 6, а).

```
101 def loop():
102     conn.setblocking(False) # неблокирующий режим работы
103     try:
104         data = conn.recv(80)   # принятие и запись сообщения с сокета
105         y = json.loads(data)    # десериализация JSON-пакета
106         # парсинг (извлечение показаний с сенсоров) и вывод в виджеты
107         p.set(y["Pressure"])
108         aX.set(y["Acceleration"][0])
109         aY.set(y["Acceleration"][1])
110         aZ.set(y["Acceleration"][2])
111         gest.set(y["Gesture"])
112     except:
113         tk.after(1, loop)   # отложенный вызов функции loop
114         return
115     tk.after(1, loop)  # отложенный вызов функции loop
116     return
```

Рисунок 29 – Функция приема и обработки входящих данных на ПК

Входящими данными являются показатели различных датчиков в виде объекта JSON. Для работы с такими объектами была использована встроенная библиотека «json» [15], позволяющая сериализовать и десериализовать JSON массивы, а также проводить их парсинг. Десериализация принятого объекта начинается после того, как сокет получил определенное количество байт, которое занимает пакет JSON вместе с пустым блоком байтов для разделения пакетов.

Далее из десериализованной JSON структуры извлекаются значения, соответствующие конкретным ключам (выполняется парсинг данных). Извлеченные показания датчиков напрямую передаются в переменные, привязанные к соответствующим виджетам. В случае отсутствия принимаемых данных или неудачи при их обработке функция приема и обработки пакетов повторяется вновь. При этом она не блокирует прочие процессы в приложении, осуществляясь параллельно им.

5.4. Обработка и передача на МК вводимых данных

Обработка и передача вводимых данных осуществляется зеркально приему и обработке получаемых пакетов, лишь с учетом того, что ввод данных необходимо сделать интуитивно понятным для пользователя.

В качестве передаваемых данных выступают сигналы включения/выключения дискретных светодиодов, подключенных к МК, и массив из выбранных цветов для элементов восьмисегментной светодиодной ленты.

Сигналы для дискретных светодиодов представляют собой логическую единицу или ноль, так что для их ввода достаточно виджета «Checkbutton», который представляет собой флажок, к которому привязана переменная. Эта переменная принимает значение 1, если флажок установлен, и 0, если он убран.

Сигналы управления светодиодами на ленте представляют собой названия выбираемых цветов (Red, Blue, Green). Однако для уменьшения объема передаваемого пакета они кодируются в числа 1,2 и 3, соответственно. Для организации пользовательского ввода использованы виджеты типа «Combobox», представляющие собой выпадающие списки с выбираемыми цветами для каждого элемента светодиодной ленты. Аналогично элементу «Checkbutton» к выбираемому значению привязаны переменные, которые преобразуются в код цвета и впоследствии собираются в массив цветов светодиодной ленты, что можно наблюдать на рис. 30.

```
128  # функция сопоставления цвета его номеру
     8 usages
129  def createColor(getColor):
130      if getColor == "красный":
131          returnColor = 1
132      elif getColor == "синий":
133          returnColor = 2
134      elif getColor == "зеленый":
135          returnColor = 3
136
137  # функции создания массива цветов для ленты
     1 usage
138  def selectLS1(event):
139      LScolors[0] = createColor(comboLS1.get())
     1 usage
140  def selectLS2(event):
141      LScolors[1] = createColor(comboLS2.get())
```

Рисунок 30 – Обработка виджетов для выбора цветов для светодиодной ленты

Функции изменения элементов массива цветов светодиодной ленты привязаны к событию изменения выбранного элемента в выпадающем списке с помощью метода «bind», аргументами к которому являются действие и выполняемая функция. Функция формирования и отправки JSON пакета также привязана к соответствующей кнопке с помощью указания соответствующего аргумента command при создании виджета «Button». Вышеописанные функции и привязка их к действиям с виджетами изображены на рис. 31.

```
155 # функция отправки JSON-пакета
    1 usage
156 def send():
157     mydict = {"LED1": statLed1.get(), "LED2": statLed2.get(), "LED3": statLed3.get(), "LEDSTRIP": LScolors}  # создаем словарь
158     datas = json.dumps(mydict)  # сериализуем словарь в JSON-структуру
159     conn.sendall(bytes(datas, encoding="ascii")) # отправка JSON-пакета на сервер
160
161 # виджет типа Button для отправки данных на сервер
162 btn = ttk.Button(text="Поменять состояние светодиодов", command=send)
163 btn.place(x=490, y=100)
164
165 # команды обработки изменений выбранных цветов для ленты светодиодов
166 comboLS1.bind("<<ComboboxSelected>>", selectLS1)
167 comboLS2.bind("<<ComboboxSelected>>", selectLS2)
168 comboLS3.bind("<<ComboboxSelected>>", selectLS3)
169 comboLS4.bind("<<ComboboxSelected>>", selectLS4)
170 comboLS5.bind("<<ComboboxSelected>>", selectLS5)
171 comboLS6.bind("<<ComboboxSelected>>", selectLS6)
172 comboLS7.bind("<<ComboboxSelected>>", selectLS7)
173 comboLS8.bind("<<ComboboxSelected>>", selectLS8)
```

Рисунок 31 – Формирование и передача на МК JSON пакета

Для создания объекта JSON сначала необходимо сформировать строку, состоящую из пар ключ/значение. Ключи – это названия светодиодов или светодиодной ленты (LED1, LED2, LED3, LEDSTRIP). Значения для светодиодов возвращаются напрямую из булевых переменных, привязанных к соответствующим виджетам. А значение для светодиодной ленты – это список кодов цветов для каждого её элемента, формируемый описанным выше способом. Сформированная строка сериализуется в JSON структуру вызовом метода «dumps».

Сериализованная структура в свою очередь преобразуется в байтовую строку с указанием кодировки ASCII, которая используется в МК ESP8266, и передается на сервер с помощью метода «sendall».

Полный код созданного приложения представлен в приложении В, а алгоритм – в приложении Г.

6. БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

6.1. Требования к ПО с точки зрения эргономики

Рассмотрим соответствие интерфейса ПО, что использовалось во время разработки, и интерфейс разработанного приложения на предмет соответствия требованиям ГОСТ Р ИСО 9241 – 100 [16]. Данный стандарт является одним из международных стандартов ISO и описывает эргономику ПО в контексте взаимодействия человека с компьютером.

Следование принципам, рекомендациям и требованиям, приведенным в ГОСТ Р ИСО 9241, позволяет предотвратить у пользователей возникновение следующих проблем:

– появление необходимости в выполнении действий, не требуемых для выполнения задачи;
– вводящая в заблуждение информация;
– недостаток информации;
– неожиданная реакция интерактивной системы;
– навигационные ограничения при использовании системы;
– неэффективное восстановление после ошибок.

Для исключения возможности возникновения вышеупомянутых проблем следует руководствоваться следующими рекомендациями:

– выводимая программным обеспечением информация должна быть распознаваема, разборчива, лаконична и понятна;
– диалог, возникающий между пользователем и интерфейсом ПО, должен быть пригодным для решения поставленных задач, для обучения и индивидуализации, а также быть управляемым, информативным и устойчивым к ошибкам;
– ПО должно иметь возможность персонализации и настройки интерфейса для упрощения взаимодействия пользователя с ним;
– зрительное взаимодействие с интерфейсом должно происходить при минимальном количестве отображаемых на дисплее цветов, определенных

соотношениях яркости и контраста изображения и оптимальных размерах знаков и элементов интерфейса. Данные требования выдвигаются с целью увеличения точности считывания информации пользователем и исключения лишнего напряжения зрительных органов.

Таким образом, зная основные требования, предъявляемые к ПО для соблюдения принципов эргономики, можно приступить к анализу соответствия использованных в ходе разработки программных обеспечений, в число которых вошли среды разработки Arduino IDE и PyCharm. Также проанализируем созданное приложение для ПК под управлением Windows.

6.2. Эргономика среды разработки Arduino IDE

Arduino IDE – это интегрированная среда разработки, предназначенная для создания и загрузки программ на Arduino-совместимые платы, а также на платы других производителей.

Среда разработки Arduino IDE включает в себя редактор кода, который выделяет разными цветами команды подключения библиотек, типы переменных, объявление постоянных, создание объектов и т.д. Также она позволяет сворачивать условия, циклы, функции и комментарии в коде. Эти функции увеличивают распознаваемость и информативность представляемой информации, а также повышают пригодность Arduino IDE для обучения. Arduino IDE позволяет скомпилировать код, проверив его на ошибки, в результате в командной строке появится информация об ошибках или их отсутствии. Также среда разработки позволяет загрузить скомпилированный код на подключенную Arduino-совместимую плату, в этом случае в результате в командной строке будет представлена информация о плате, например, объем флеш-памяти, занятый загруженной программой. Снимок экрана типичного окна программы представлен на рис. 32.

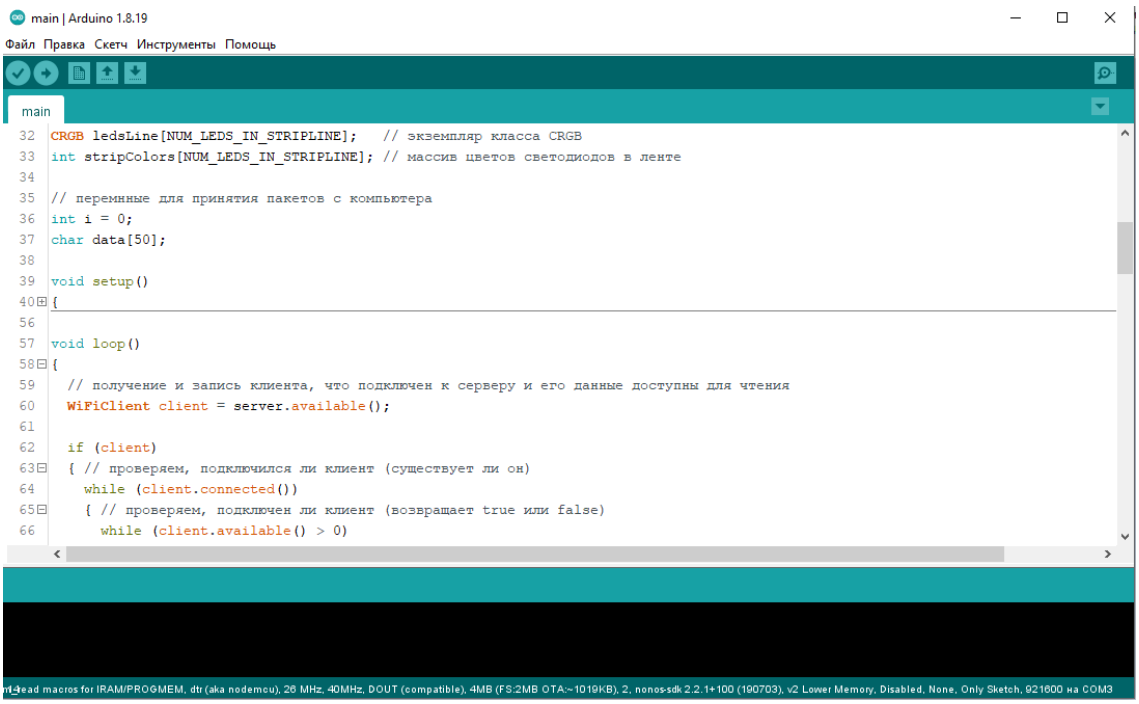


Рисунок 32 – Снимок экрана программы Arduino IDE

Одним из главных достоинств рассматриваемой среды разработки является встроенный менеджер библиотек. Он изображен на рис. 33. С помощью него можно найти необходимую библиотеку для подключения совместимого микроконтроллера или управления подключаемым к микроконтроллеру устройством. Каждая библиотека включает в себя множество примеров использования встроенных в неё функций.

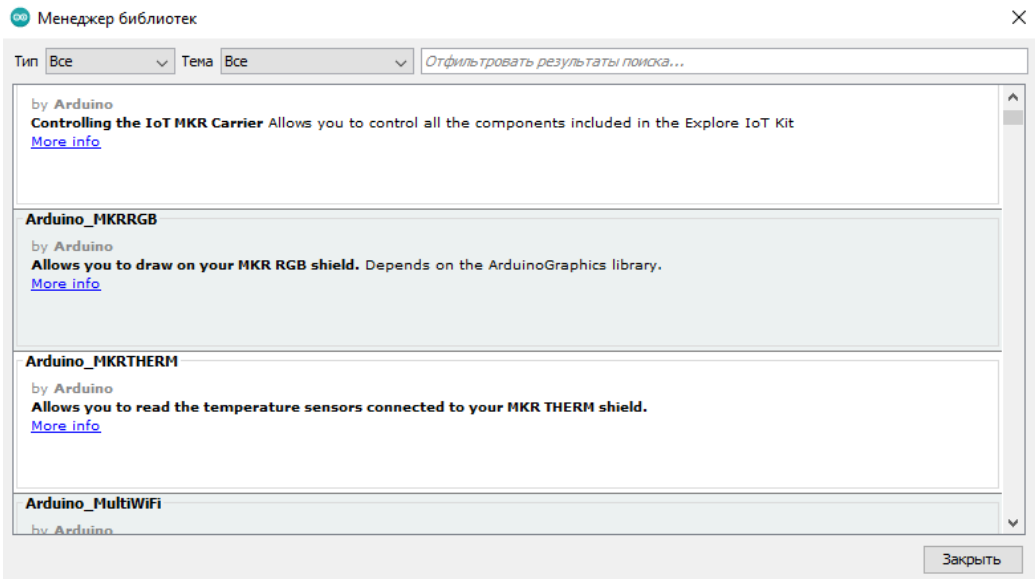


Рисунок 33 – Менеджер библиотек в Arduino IDE

Также стоить отметить возможность вносить изменения в режимы работы подключенной платы, что приведено на рис. 34. Например, возможность выбора частоты работы процессора и флеш-памяти. Данная функция среды разработки Arduino IDE повышает пригодность рассматриваемого ПО для решения различных задач.

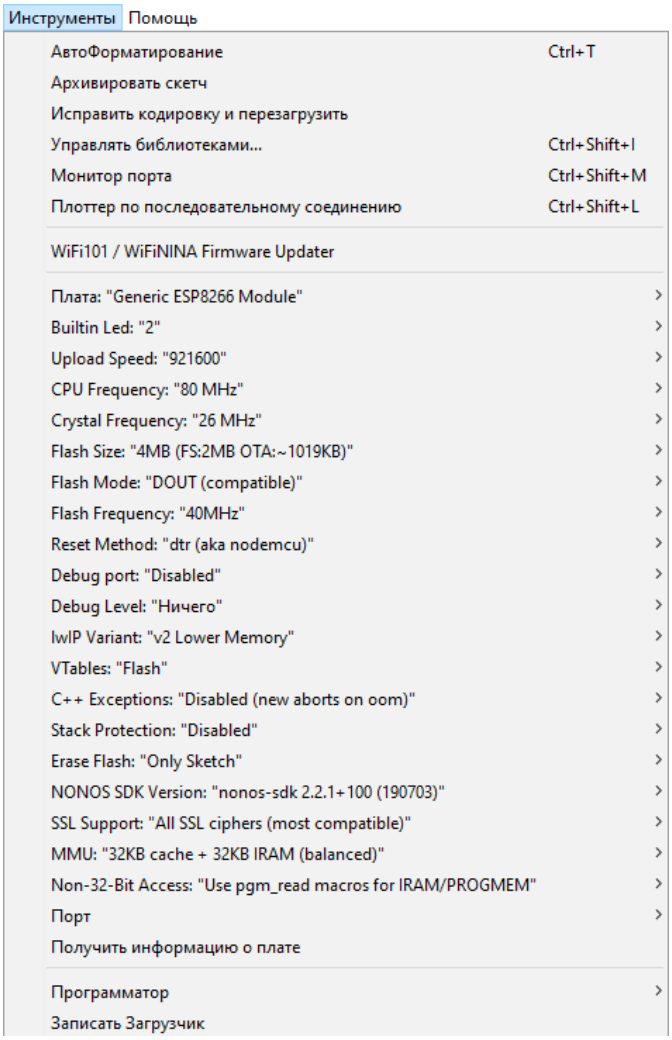


Рисунок 34 – Настройки подключенной платы в Arduino IDE

Итак, среда разработки Arduino IDE удовлетворяет большинству требований, призванных повысить её эргономику. Работа с этой средой создает обширные возможности в условиях реальной практической деятельности, которая проводилась во время разработки программы для управления МК ESP8266.

6.3. Эргономика среды разработки PyCharm

PyCharm – это интегрированная среда разработки для языка программирования Python [17]. Предоставляет пользователю комплекс средств для написания кода и визуальный отладчик. Типичное окно программы представлено на рис. 35.

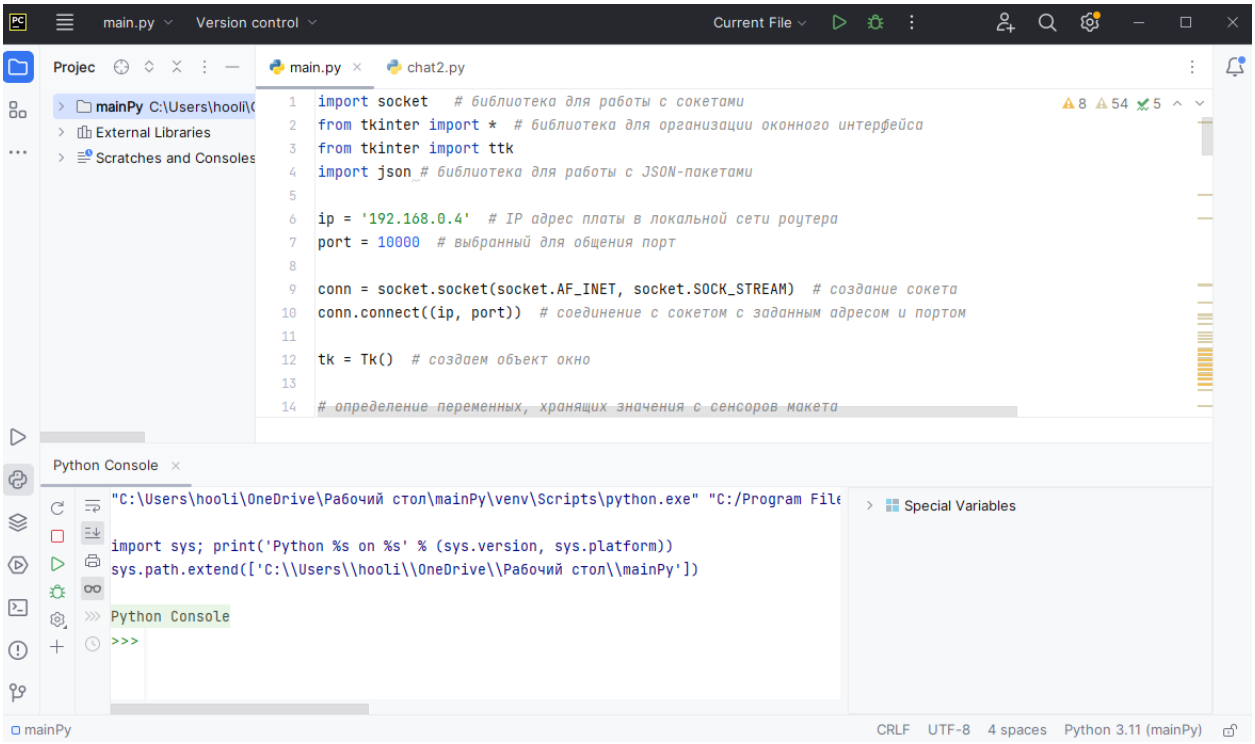


Рисунок 35 – Снимок экрана программы PyCharm

Редактор кода кроме функций, аналогичным ранее описанным функциям среды Arduino IDE, таких как подцветка синтаксиса, обладает отличительными особенностями в виде автодополнения и рефакторинга кода. Технология автодополнения выводит подсказки в виде возможных функций, переменных и объектов, при вводе первых букв слова. Также при выборе объявляемого объекта можно наблюдать подсказки, несущие информацию о среде разработки или синтаксисе языка Python. Рефакторинг кода – это переработка исходного кода программы без изменения выполняемых им функций с целью его упрощения и облегчения визуального восприятия. Данные функции позволяют создать диалог между интерфейсом и пользователем, который призван сделать ПО более информативным и устойчивым к ошибкам.

Среда разработки PyCharm имеет большое количество различных окон, которые предназначены для отладки программы, общения с помощью консоли, управления файлами в проекте, создание выполняемых файлов из созданных программ и т.д. Взаимное расположение всех окон можно менять, их можно удалять, менять их цветовую гамму. Так, интерфейс среды PyCharm имеет большие возможности индивидуализации, персонализации и настройки.

Таким образом, среда разработки PyCharm полностью удовлетворяет эргономическим требованиям и рекомендациям. Она обладает намного большим функционалом чем рассмотренная ранее Arduino IDE, так как язык программирования Python более распространен и применяется для большего спектра задач. Например, организация сетевого взаимодействия, обработка данных и создание приложений, ведь все эти задачи выполнялись при разработке приложения для приема и обработки данных с МК на ПК.

6.4. Эргономика разработанного приложения

Разработанный в ходе выполнения работы пользовательский интерфейс предназначен для вывода данных с датчиков, и ввода данных для управления светодиодами. Снимок экрана приложения приведен на рис. 36.

Принимаемые данные сгруппированы в левой части окна приложения под соответствующим заголовком. Для каждого датчика указана измеряемая величина и её единицы измерения. Данные свойства интерфейса призваны повысит разборчивость и лаконичность представляемой информации.

Пользователь не имеет возможности изменить выводимые показания датчиков, кроме того, эти показания постоянно обновляются. Так образом повышается устойчивость ПО к ошибкам.

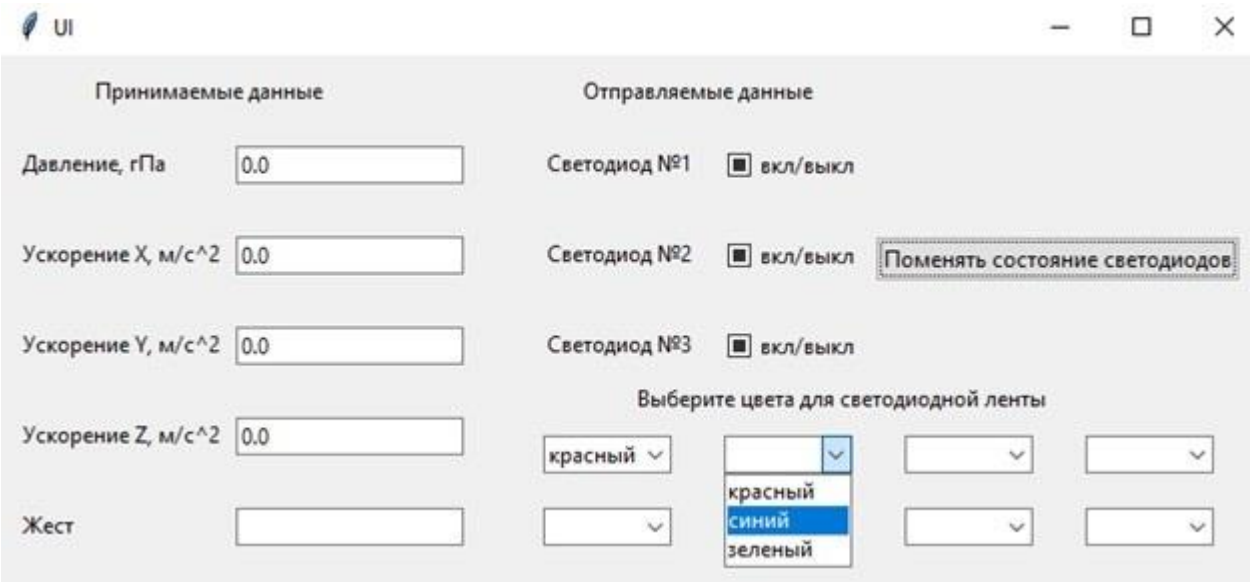


Рисунок 36 – Снимок экрана разработанного приложения

Аналогично принимаемым данным отправляемые данные представлены в правой части окна в целях соблюдения лаконичности интерфейса.

Для включения и выключения дискретных светодиодов используется флажок, а для управления цветами светодиодной ленты выпадающий список. Данные способы ввода информации использованы для исключения возможных ошибок при работе с приложением, то есть для повышения устойчивости к ошибкам.

Подводя итог, можно утверждать, что разработанный пользовательский интерфейс соответствует большинству эргономических рекомендаций. Однако в нем отсутствуют возможности настройки интерфейса, которые не были реализованы из-за простоты ПО, которое предназначено для использования в лаборатории кафедры РЭС.

ЗАКЛЮЧЕНИЕ

В данной работе были изучены принципы построения сетевых структур клиент-сервер. Полученные навыки были применены для построения беспроводной связи в локальной сети Wi-Fi между макетом под контролем МК ESP8266 и приложением на ПК.

Проанализировав вычислительные мощности МК и ПК, была использована многоуровневая модель клиент-сервер с равномерным распределением нагрузки между МК и ПК. В целях обеспечения стабильности соединения и точности передаваемых данных использовался сетевой протокол TCP для передачи данных между клиентом и сервером.

В целях демонстрации возможностей МК ESP8266 и среды программирования Arduino IDE был создан макет-сервер, включающий в себя различные источники информации и приемники. Для дистанционного управления макетом с ПК под руководством Windows было создано клиентское приложение на Python, которое обменивается данными в виде текстовых JSON пакетов с МК по локальной беспроводной сети Wi-Fi.

В дополнение, была произведена оценка эргономики разработанного приложения и сред разработки, что использовались во время выполнения данной работы.

Созданное ПО может использоваться для создания лабораторных работ на кафедре РЭС напрямую или с предварительным увеличением функционала. Также оно может служить основой для построения систем, следующих концепции IOT, например, для умного дома.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Интернет вещей с ESP8266. Марко Шварц. СПб.: БХВ-Петербург, 2018. 192 с.

2. Многоуровневые системы клиент-сервер. [Электронный ресурс]. URL: https://www.osp.ru/nets/1997/06/142618 (дата обращения: 21.05.2023)

3. UNIX: разработка сетевых приложений. У.Р. Стивенс, Б. Феннер, Э.М. Рудофф. СПб.: Изд-во Питер, 2007. 1039с.

4. Сети TCP/IP, том 3. Разработка приложений типа клиент/сервер для Linux/POSIX. Камер, Дуглас Э., Стивенс, Дэвид Л. М.: Издательский дом «Вильямс», 2002. 592 с.

5. ESP8266EX. Datasheet. Espressif Systems, 2023. 31 с.

6. BMP180 Data sheet. Bosch Sensortec, 2013. 28 с.

7. ADX-354 datasheet. [Электронный ресурс]. URL: https://ru.mouser.com/datasheet/2/609/adxl354_355-1503915.pdf (дата обращения: 21.05.2023)

8. APDS9960 datasheet. [Электронный ресурс]. URL: https://content.arduino.cc/assets/Nano_BLE_Sense_av02-4191en_ds_apds9960.pdf (дата обращения: 21.05.2023)

9. Arduino Software. [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Arduino (дата обращения: 21.05.2023)

10. Документация к библиотеке WiFi в Arduino IDE. [Электронный ресурс]. URL: https://www.arduino.cc/reference/en/libraries/wifi/ (дата обращения: 21.05.2023)

11. Введение в JSON. [Электронный ресурс]. URL: https://www.json.org/json-ru.html (дата обращения: 21.05.2023)

12. Mastering ArduinoJson 6. [Электронный ресурс]. URL: https://arduinojson.org/book/deserialization_tutorial6.pdf (дата обращения: 21.05.2023)

13. Документация к библиотеке tkinter в Python. [Электронный ресурс]. URL: https://docs.python.org/3/library/tkinter.html (дата обращения: 21.05.2023)

14. Документация к библиотеке socket в Python. [Электронный ресурс]. URL: https://docs.python.org/3/library/socket.html (дата обращения: 21.05.2023)

15. Документация к библиотеке json в Python. [Электронный ресурс]. URL: https://docs.python.org/3/library/json.html (дата обращения: 21.05.2023)

16. ГОСТ Р 55241.1-2012/ISO/TR 9241-100:2010 Эргономика взаимодействия человек-система. Часть 100. Введение в стандарты, относящиеся к эргономике программных средств

17. Описание среды разработки PyCharm. Электронный ресурс]. URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата обращения: 21.05.2023)

Приложение А

Код программы управления МК

```
#include <ESP8266WiFi.h>    // библиотека для создания и работы с Wi-Fi сервером
#include "ArduinoJson.h"    // библиотека для работы с JSON пакетами
#include <Wire.h>   // библиотека для связи с датчиками через интерфейс I2C
#include <SFE_BMP180.h>   // библиотека для управления барометром
#include <Adafruit_Sensor.h>    // библиотека для датчиков Adafruit
#include <Adafruit_ADXL345_U.h>   // библиотека для управления акселерометром
#include <SparkFun_APDS9960.h>    // библиотека для работы с датчиком жестов
#include <FastLED.h>    // библиотека для управления светодиодной лентой

SFE_BMP180 pressure;  // создание экземпляра класса SFE_BMP180 для работы с барометром
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345); // создание объекта, экземпляр класса ADXL345_U для работы с акселерометром
SparkFun_APDS9960 apds = SparkFun_APDS9960(); // создание объекта, экземпляр класса SparkFun_APDS9960 для работы с датчиком жестов
int gesture;

const char* ssid = "RT-GPON-B750";   // SSID
const char* password = "HFmk4UdK";  // пароль
WiFiServer server(10000); // создание сервера, что общается по указанному порту

//объявление констант (названий выходам 12 (D6), 13 (D7), 15 (D8))
#define PIN_LED1 12
#define PIN_LED2 13
#define PIN_LED3 15
#define NUM_LEDS_IN_STRIPLINE 8   // количество светодиодов на ленте
#define DATA_PIN 16   // пин для подключения ленты
CRGB ledsLine[NUM_LEDS_IN_STRIPLINE];   // экземпляр класса CRGB
int stripColors[NUM_LEDS_IN_STRIPLINE]; // массив цветов светодиодов в ленте

// переменые для принятия пакетов с компьютера
int i = 0;
char data[70];

void setup()
{
  WiFi.begin(ssid, password);    // поключение к точке доступа
  server.begin();   // сервер начинает "слушать" входящие запросы
  // настройка датчика жестов
  apds.init();
  apds.enableGestureSensor(true);
  // перевод цифровых портов со светодиодами в режим вывода
  pinMode(PIN_LED1, OUTPUT);
  pinMode(PIN_LED2, OUTPUT);
  pinMode(PIN_LED3, OUTPUT);
  // настройка светодиодной ленты
```

```
  FastLED.addLeds <WS2812, DATA_PIN, GRB>(ledsLine,
NUM_LEDS_IN_STRIPLINE).setCorrection(TypicalLEDStrip);
  FastLED.setBrightness(50);
}

void loop()
{
  // получение и запись клиента
  WiFiClient client = server.available();
// проверка, существует ли клиент
  if (client)
  {
    while (client.connected())  // проверка, подключен ли клиент
    {
      while (client.available() > 0)  // проверка, есть ли байты для чтения от клиента
      {
        data[i] = static_cast<char>(client.read()); // считываем и записываем первый на очереди
байт
        i++;
      }
      DynamicJsonDocument parsed(1024); // выделение памяти для принятой JSON строки
      deserializeJson(parsed, data);  // десериализация принятого JSON пакета
      //парсинг пакета JSON
      // получение сигналов на дискретные светодиоды
      int LED1_status = parsed["LED1"];
      int LED2_status = parsed["LED2"];
      int LED3_status = parsed["LED3"];
      // получение массива сигналов на светодиодную ленту
      for (int i = 0; i < NUM_LEDS_IN_STRIPLINE - 1; i++)
      {
        stripColors[i] = parsed["LEDSTRIP"][i];  // запись цветов светодиодов в ленте
      }
      // смена цветов дискретных светодиодов
      digitalWrite(PIN_LED1, LED1_status);
      digitalWrite(PIN_LED2, LED2_status);
      digitalWrite(PIN_LED3, LED3_status);
      // смена цветов светодиодов в ленте
      for (int a = 0; a < NUM_LEDS_IN_STRIPLINE; a++)
      {
        changeLedInStrip(a, stripColors[a]);
      }
// выделение памяти для отправляемого JSON объекта
      DynamicJsonDocument doc(64);

      double Pr = getPressure();  // измерение давления
// создание структуры для занесения в неё показаний акселерометра
      sensors_event_t event;
// получение текущих показаний акселерометра
      accel.getEvent(&event);
      double aX = event.acceleration.x;
      double aY = event.acceleration.y;
      double aZ = event.acceleration.z;
```

```
// получение текущих показаний датчика жестов
      gesture = apds.readGestur();

// создание пар ключ-значение (сенсор-показания) в структуре JSON
      doc["Pressure"] = Pr; // давление в гПа
      doc["Acceleration"][0] = aX;  // ускорение X в м/с^2
      doc["Acceleration"][1] = aY;  // ускорение Y в м/с^2
      doc["Acceleration"][0] = aZ;  // ускорение Z в м/с^2
      doc["Gesture"][0] = gesture;  // жест
      String output;  // создание переменной для хранения JSON пакета
      serializeJson(doc, output);   // запись JSON пакета в ранее созданную переменную
      for (int i = 0; i < output.length(); i++)
      {
        client.write(output[i]);
      }
      client.write("    ");
    }
  }
}
// функция измерения давления
double getPressure()
{
  char status;
  double T, P;
  pressure.startTemperature(); // запуск измерения температуры
  delay(status);  // ожидание завершения измерений
  pressure.getTemperature(T);  // извлечение данных о температуре
  pressure.startPressure(3);  // запуск измерения давления
  delay(status);
  pressure.getPressure(P, T); // извлечение данных о давлении
  return (P);   //возвращение измеренного значения в мбар (гПа)
}
// функция смены цветов светодиодной ленты
void changeLedInStrip(int ledPos, int ledColor)
{
  switch (ledColor)
  {
    case 1:
      ledsLine[ledPos] = CRGB::Red;
      FastLED.show();
      break;
    case 2:
      ledsLine[ledPos] = CRGB::Green;
      FastLED.show();
      break;
    case 3:
      ledsLine[ledPos] = CRGB::Blue;
      FastLED.show();
      break;
  }
}
```

Приложение Б

Алгоритм программы управления МК

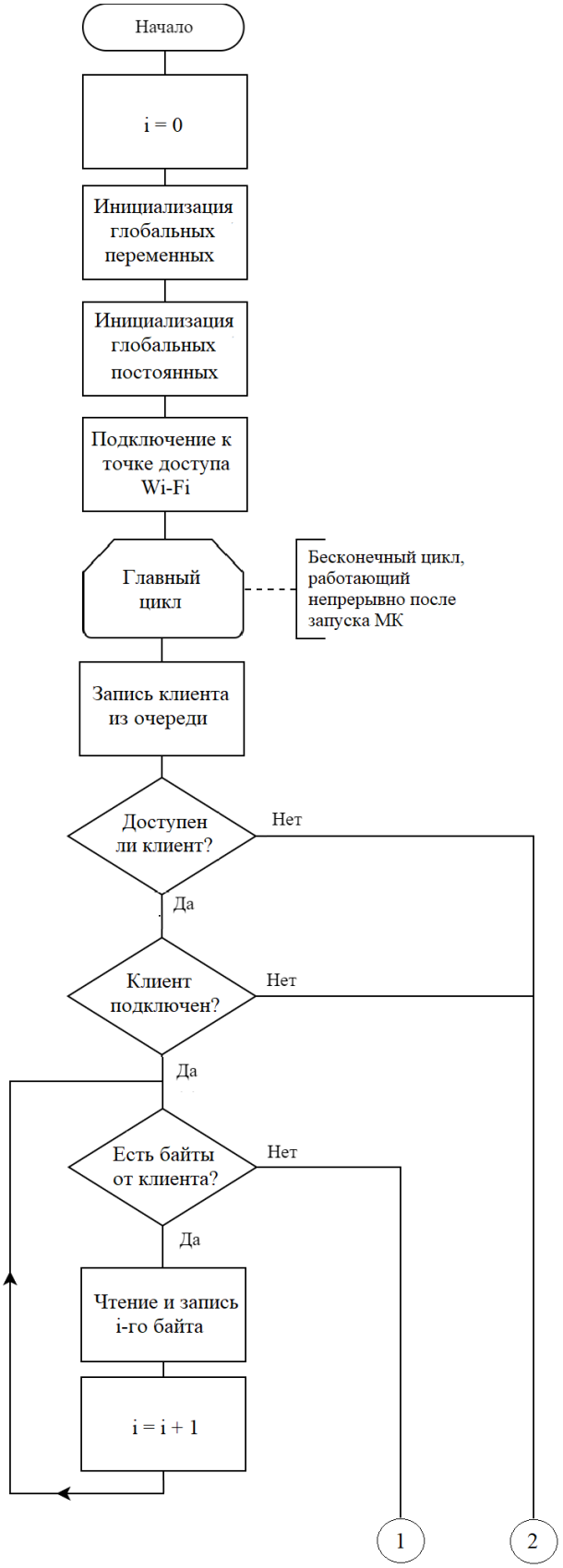
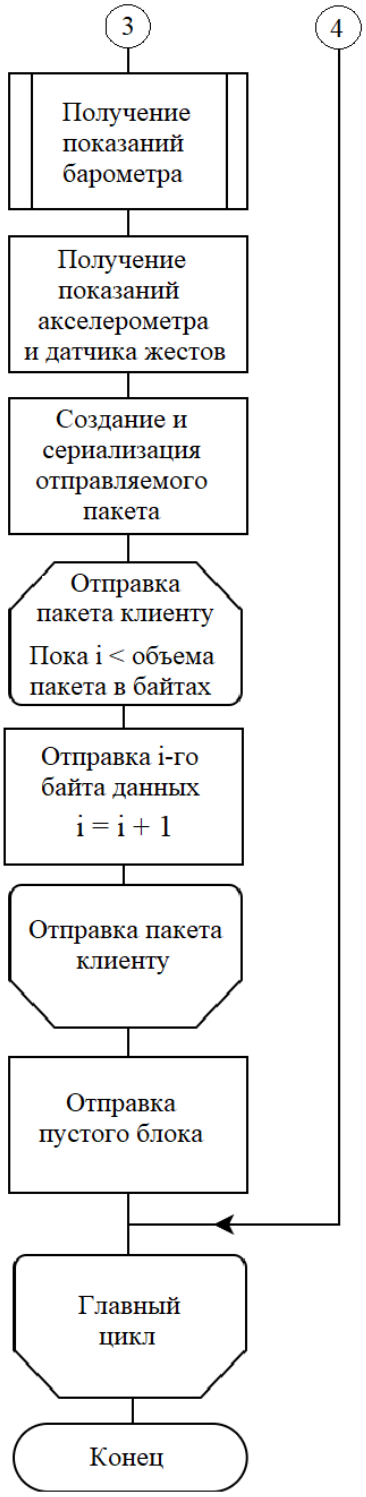


Рисунок Б.1 – Алгоритм программы управления МК

Продолжение рисунка Б.1

Продолжение рисунка Б.1



Приложение В

Код клиентского приложения для ПК

```
import socket   # библиотека для работы с сокетами
from tkinter import *  # библиотека для организации оконного интер-
фейса
from tkinter import ttk
import json # библиотека для работы с JSON-пакетами

ip = '192.168.0.4'  # IP адрес МК в локальной сети роутера
port = 10000  # выбранный для общения порт

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # создание
сокета
conn.connect((ip, port))  # соединение с сокетом с заданным адресом и
портом

tk = Tk()  # создание объекта окна

# определение переменных, хранящих значения с сенсоров макета
p = DoubleVar()
aX = DoubleVar()
aY = DoubleVar()
aZ = DoubleVar()
gest = StringVar()

# массив возможных цветов светодиодов на ленте
ledStripColors = ["красный", "синий", "зеленый"]

# список значений для ленты светодиодов, передаваемый в JSON-пакет
LScolors = [1, 2, 3, 1, 2, 3, 1, 2]

# определение переменных, хранящих состояния дискретных светодиодов
statLed1 = IntVar()
statLed2 = IntVar()
statLed3 = IntVar()

tk.title('UI')  # заголовок окна
tk.geometry('700x300')  # размеры окна

# создание виджетов типа Entry для вывода показаний датчиков
valP = ttk.Entry(textvariable=p)
valAx = ttk.Entry(textvariable=aX)
valAy = ttk.Entry(textvariable=aY)
valAz = ttk.Entry(textvariable=aZ)
valGest = ttk.Entry(textvariable=gest)

# создание виджетов типа Label для текстовых меток
labelRecieved = ttk.Label(text="Принимаемые данные")
labelP = ttk.Label(text="Давление, гПа")
labelAx = ttk.Label(text="Ускорение X, м/с^2")
labelAy = ttk.Label(text="Ускорение Y, м/с^2")
labelAz = ttk.Label(text="Ускорение Z, м/с^2")
labelG= ttk.Label(text="Жест")
labelSend = ttk.Label(text="Отправляемые данные")
labelLed1 = ttk.Label(text="Светодиод №1")
```

```
labelLed2 = ttk.Label(text="Светодиод №2")
labelLed3 = ttk.Label(text="Светодиод №3")
labelLedS = ttk.Label(text="Выберите цвета для светодиодной ленты")

# создание виджетов типа Checkbutton для управления светодиодами
btnL1 = ttk.Checkbutton(text="вкл/выкл", variable=statLed1)
btnL2 = ttk.Checkbutton(text="вкл/выкл", variable=statLed2)
btnL3 = ttk.Checkbutton(text="вкл/выкл", variable=statLed3)

# создание виджетов типа Combobox для управления светодиодной лентой
comboLS1 = ttk.Combobox(values=ledStripColors, width=8,
state="readonly")
comboLS2= ttk.Combobox(values=ledStripColors, width=8,
state="readonly")
comboLS3 = ttk.Combobox(values=ledStripColors, width=8,
state="readonly")
comboLS4 = ttk.Combobox(values=ledStripColors, width=8,
state="readonly")
comboLS5 = ttk.Combobox(values=ledStripColors, width=8,
state="readonly")
comboLS6 = ttk.Combobox(values=ledStripColors, width=8,
state="readonly")
comboLS7 = ttk.Combobox(values=ledStripColors, width=8,
state="readonly")
comboLS8 = ttk.Combobox(values=ledStripColors, width=8,
state="readonly")

# размещение виджетов в окне
labelRecieved.place(x=50, y=10)
labelP.place(x=10, y=50)
labelP.place(x=10, y=50)
labelAx.place(x=10, y=100)
labelAy.place(x=10, y=150)
labelAz.place(x=10, y=200)
labelG.place(x=10, y=250)
valP.place(x=130, y=50)
valAx.place(x=130, y=100)
valAy.place(x=130, y=150)
valAz.place(x=130, y=200)
valGest.place(x=130, y=250)
labelSend.place(x=320, y=10)
labelLed1.place(x=300, y=50)
labelLed2.place(x=300, y=100)
labelLed3.place(x=300, y=150)
labelLedS.place(x=350, y=180)
btnL1.place(x=400, y=50)
btnL2.place(x=400, y=100)
btnL3.place(x=400, y=150)
comboLS1.place(x=300, y=210)
comboLS2.place(x=400, y=210)
comboLS3.place(x=500, y=210)
comboLS4.place(x=600, y=210)
comboLS5.place(x=300, y=250)
comboLS6.place(x=400, y=250)
comboLS7.place(x=500, y=250)
comboLS8.place(x=600, y=250)
```

```
# функция loop, обрабатывающая входящие JSON-пакеты
def loop():
    conn.setblocking(False) # неблокирующий режим работы
    try:
        data = conn.recv(80)    # принятие и запись сообщения с сокета
        y = json.loads(data)     # десериализация JSON-пакета
        # парсинг (извлечение показаний с сенсоров) и вывод в виджеты
        p.set(y["Pressure"])
        aX.set(y["Acceleration"][0])
        aY.set(y["Acceleration"][1])
        aZ.set(y["Acceleration"][2])
        gest.set(y["Gesture"])
    except:
        tk.after(1, loop)    # отложенный вызов функции loop
        return
    tk.after(1, loop)  # отложенный вызов функции loop
    return
# функция отправки JSON-пакета
def send():
    mydict = {"LED1": statLed1.get(), "LED2": statLed2.get(), "LED3":
statLed3.get(), "LEDSTRIP": LScolors}  # создание словаря
    datas = json.dumps(mydict)  # сериализация словаря в JSON-струк-
туру
    conn.sendall(bytes(datas, encoding="ascii")) # отправка JSON-па-
кета на сервер

# виджет типа Button для отправки данных на сервер
btn = ttk.Button(text="Поменять состояние светодиодов", command=send)
btn.place(x=490, y=100)

# функция сопоставления цвета его номеру
def createColor(getColor):
    if getColor == "красный":
        returnColor = 1
    elif getColor == "синий":
        returnColor = 2
    elif getColor == "зеленый":
        returnColor = 3
# функции создания массива цветов для ленты
def selectLS1(event):
    LScolors[0] = createColor(comboLS1.get())
def selectLS2(event):
    LScolors[1] = createColor(comboLS2.get())
def selectLS3(event):
    LScolors[2] = createColor(comboLS3.get())
def selectLS4(event):
    LScolors[3] = createColor(comboLS4.get())
def selectLS5(event):
    LScolors[4] = createColor(comboLS5.get())
def selectLS6(event):
    LScolors[5] = createColor(comboLS6.get())
def selectLS7(event):
    LScolors[6] = createColor(comboLS7.get())
def selectLS8(event):
    LScolors[7] = createColor(comboLS8.get())

# команды обработки изменений выбранных цветов для ленты светодиодов
```

```
comboLS1.bind("<<ComboboxSelected>>", selectLS1)
comboLS2.bind("<<ComboboxSelected>>", selectLS2)
comboLS3.bind("<<ComboboxSelected>>", selectLS3)
comboLS4.bind("<<ComboboxSelected>>", selectLS4)
comboLS5.bind("<<ComboboxSelected>>", selectLS5)
comboLS6.bind("<<ComboboxSelected>>", selectLS6)
comboLS7.bind("<<ComboboxSelected>>", selectLS7)
comboLS8.bind("<<ComboboxSelected>>", selectLS8)

tk.after(1, loop)  # отложенный вызов функции loop
tk.mainloop()  # запуск цикла обработки событий окна
```

Приложение Г

Алгоритм клиентского приложения для ПК

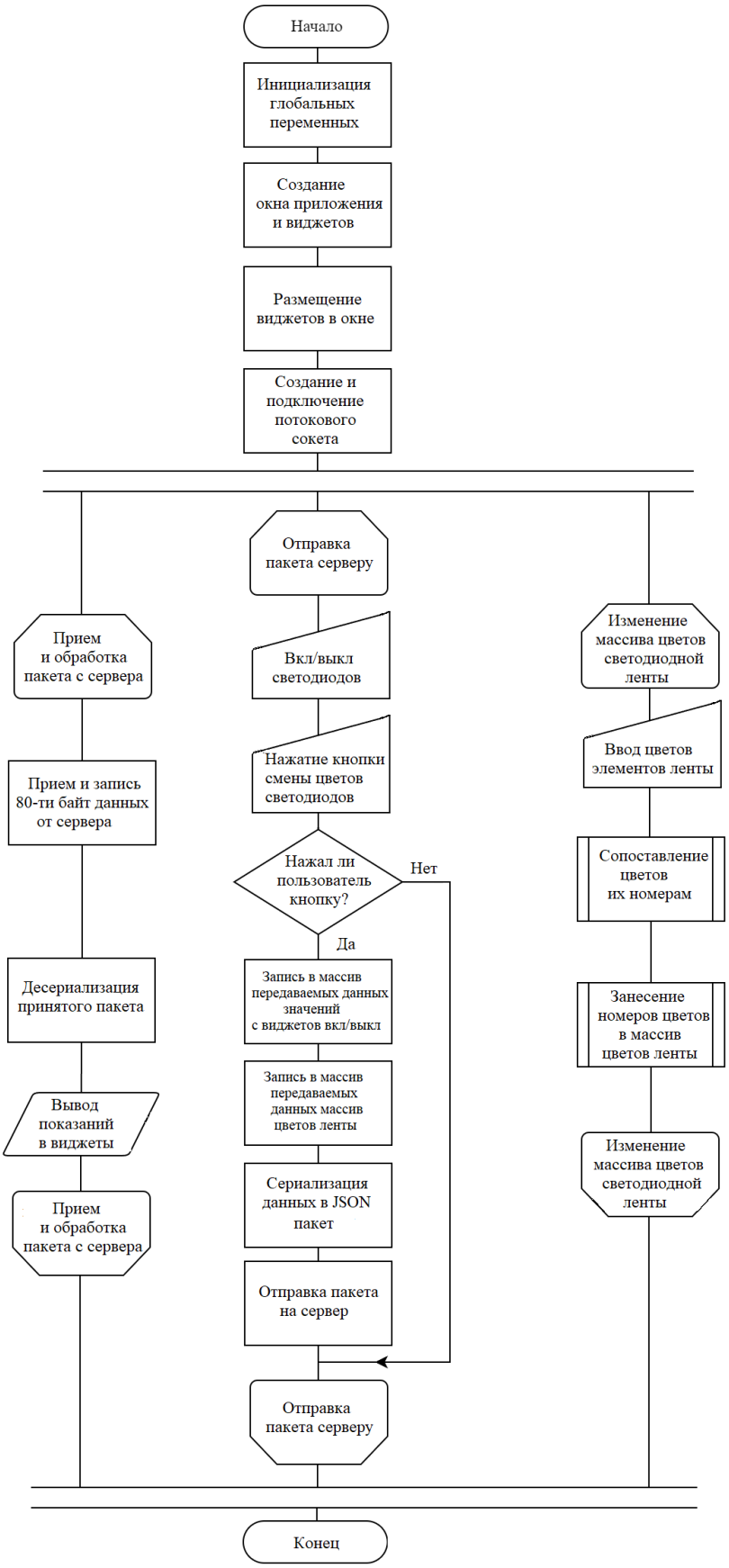


Рисунок Г.1 – Алгоритм клиентского приложения для ПК