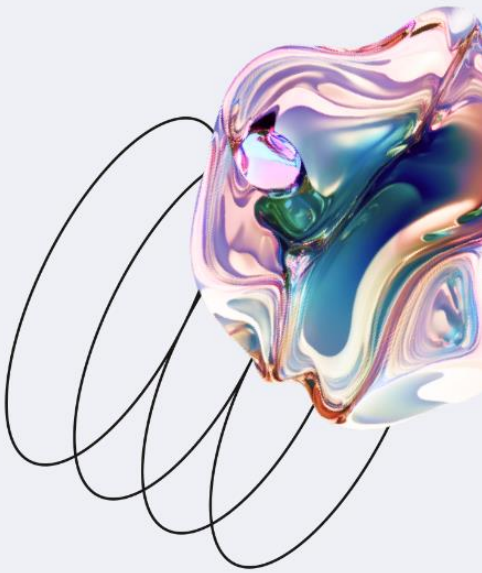
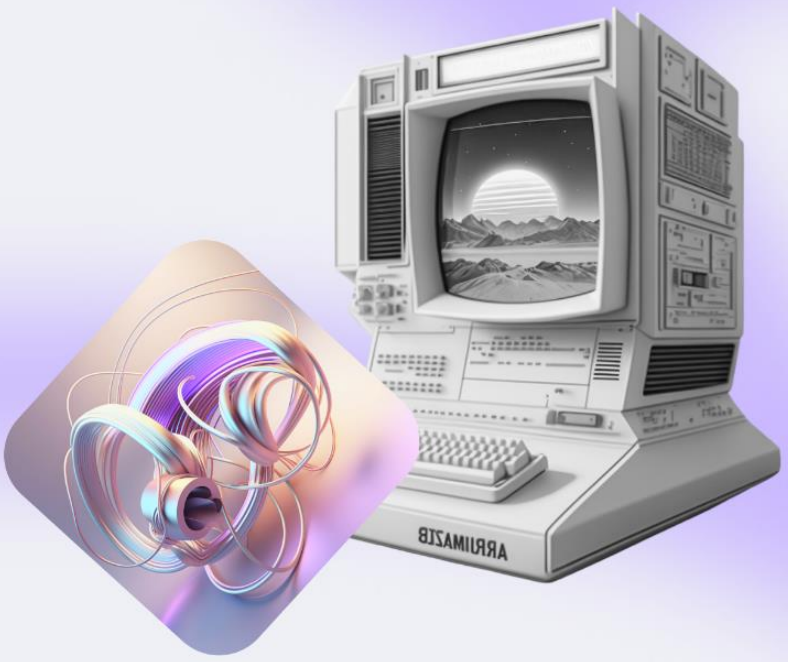
GeekBrains



Дипломная работа по курсу «Кибернетика умных устройств (IoT)»

Бурцев А.Б.



Оглавление

Постановка задания

Создание клиент-серверной платформы — «Статистика температуры»

Цель проекта: Разработать клиент-серверную платформу на базе оконечного оборудования Raspberry Pi для измерения температуры с использованием датчика DS18B20, считывание данных в файл .csv, организовать передачу данных в среду Node-Red с помощью протокола MQTT и сохранение данных в .csv файл на сервере.

Какую проблему решает проект:
Проект решает проблему создания эффективной и автоматизированной системы для измерения температуры и сохранения данных в удобном формате для последующего анализа и обработки.

По какой специализации вы будете делать дипломный проект:
Специализация: Кибернетика умных устройств и систем на базе IoT-технологий и микроконтроллеров.

Есть ли у вас полезный опыт для решения этой задачи:
В качестве опыта, я имею знания и навыки в разработке программного обеспечения на языке C, использовании Raspberry Pi, Debian, работе с датчиками и интеграции с IoT-платформами.

Какими инструментами вы будете пользоваться:

1. Raspberry Pi.
2. Датчик температуры DS18B20.
3. C для программирования клиентской части на Raspberry Pi.
4. Библиотеки wiringPi и OneWire для работы с датчиком DS18B20.
5. Библиотека paho-mqtt для работы с протоколом MQTT.
6. Node-Red для организации серверной части и обработки данных.
7. Git для управления версиями проекта.

Какие технологии вы планируете использовать при выполнении проекта:

1. Протокол MQTT для передачи данных с Raspberry Pi на сервер.
2. Node-Red для организации серверной части и обработки данных.
3. Файловый формат .csv для хранения и обработки данных на сервере.

Состав команды:
Разработчик программного обеспечения

Краткое содержание структурных частей.

Теоретическая часть:

1. Описание основных принципов работы Raspberry Pi.
2. Введение в датчики температуры и их использование в системах автоматизации.
3. Обзор протокола MQTT и его применение в IoT-системах.
4. Описание Node-Red и его возможностей для обработки данных.
5. Введение в формат .csv и его использование для хранения данных.

Гипотеза:

Предположение о возможности создания клиент-серверной платформы.

Алгоритм работы клиент-серверной платформы:

Включает основные шаги по созданию и настройке системы.

Практическая часть:

1. Описание подключения датчика DS18B20 к Raspberry Pi.
2. Написание программы на C для измерения температуры с использованием датчика DS18B20.
3. Настройка Raspberry Pi для передачи данных с использованием протокола MQTT.
4. Создание топика MQTT для передачи данных температуры.
5. Настройка Node-Red для приёма данных из топика MQTT и сохранения данных в .csv файл на сервере.
6. Тестирование и отладка системы.
7. Описание результатов работы системы и анализ полученных данных.

Заключение:

1. Краткие теоретические и практические выводы, полученные во время разработки клиент-серверной платформы на базе Raspberry Pi для измерения температуры датчиком DS18B20, сохранением данных в .csv файл, передачей данных с помощью протокола MQTT на сервер в среду Node-Red и последующим сохранением данных на сервер в .csv файл.
2. Оценка проведённого исследования и достигнутых результатов.
3. Практическая значимость работы и рекомендации по совершенствованию системы.

4. Описание достижения цели проекта, выполнение задач и доказательство гипотезы.
5. Предложения по совершенствованию системы для измерения температуры и обработки данных.

Теоретическая часть:

1. Описание основных принципов работы Raspberry Pi:
   Raspberry Pi - это одноплатный компьютер, разработанный фондом Raspberry Pi. Он предназначен для обучения и изучения информатики и является одним из самых популярных одноплатных компьютеров на рынке. Основные характеристики Raspberry Pi включают:

   Процессор ARM: Raspberry Pi оснащен процессором ARM, который обеспечивает высокую производительность и энергоэффективность. Процессоры ARM являются одними из самых распространенных процессоров в мире, используемых в мобильных устройствах, таких как смартфоны и планшеты.

   Память: Raspberry Pi оснащен оперативной памятью, которая позволяет ему выполнять различные задачи. Количество памяти зависит от модели Raspberry Pi, например, Raspberry Pi 4 имеет до 8 ГБ оперативной памяти.

   Порты ввода-вывода: Raspberry Pi имеет различные порты ввода-вывода, такие как USB, HDMI, Ethernet и GPIO (General Purpose Input/Output), которые позволяют подключать к нему различные устройства, такие как клавиатуры, мыши, мониторы, сетевые кабели и другие периферийные устройства.

   Операционная система: Raspberry Pi работает под управлением операционной системы, которая может быть установлена на SD-карту или microSD-карту. Операционная система может быть выбрана из различных дистрибутивов Linux, таких как Raspbian, Ubuntu MATE, Fedora и других.

   Языки программирования: Raspberry Pi поддерживает работу с различными языками программирования, такими как Python, C, C++, Java, Scratch и другими. Python является самым популярным языком программирования для Raspberry Pi, и многие проекты и библиотеки для Raspberry Pi написаны на Python.

   Raspberry Pi может использоваться для различных задач, таких как создание домашних автоматизаций, робототехники, игровых консолей, медиацентров, серверов и многого другого. Он также поддерживает работу с различными протоколами и технологиями, такими как MQTT, Node-Red, GPIO, I2C, SPI и другими.

   Важным аспектом работы с Raspberry Pi является знание основ электроники и программирования, а также понимание принципов работы операционных систем и сетевых технологий.

2. Введение в датчики температуры и их использование в системах автоматизации:
   Датчики температуры - это устройства, предназначенные для измерения температуры окружающей среды. Они используются в различных областях, таких как промышленность, медицина, научные исследования, автоматизация и других. Датчики температуры могут быть различных типов, таких как термометры, термопара, оптические датчики температуры и другие. Один из самых популярных датчиков температуры - это датчик DS18B20.

   Датчик DS18B20 - это датчик температуры, основанный на технологии 1-Wire. Он может измерять температуру в диапазоне от -55 до +125 градусов Цельсия с точностью до 0,5 градуса Цельсия. Датчик DS18B20 имеет высокую чувствительность и быстродействие, что делает его хорошим выбором для многих приложений.

   Датчик DS18B20 может быть использован в системах автоматизации для контроля температуры в помещении, в системах охлаждения и отопления, в системах контроля качества воздуха и других системах. Он может быть подключен к Raspberry Pi или другим одноплатным компьютерам через интерфейс GPIO, а также через USB или Ethernet.

   Датчик DS18B20 может работать в режиме сбора данных, когда он измеряет температуру постоянно и передает данные на сервер или другое устройство. Он также может работать в режиме оповещения, когда он отправляет сигнал, если температура превышает или опускается ниже заданного значения.

   В системах автоматизации датчик DS18B20 может быть использован вместе с другими датчиками и актуаторами для создания полноценной системы управления. Например, он может быть использован вместе с датчиками влажности, освещенности и движения для создания системы охлаждения и отопления, которая автоматически регулирует температуру в зависимости от потребностей.

   В процессе работы с датчиками температуры важно учитывать их точность, чувствительность, быстродействие, а также условия эксплуатации. Необходимо также учитывать безопасность и надежность систем, в которых используются датчики температуры, и обеспечить их защиту от несанкционированного доступа и воздействия внешних факторов.

3. Обзор протокола MQTT и его применение в IoT-системах:
   MQTT (Message Queuing Telemetry Transport) - это протокол передачи

данных, который используется для обмена сообщениями между устройствами в сети. Он был разработан в 1999 году компанией IBM для использования в системах мониторинга и управления в реальном времени. MQTT основан на архитектуре клиент-сервер и поддерживает публикацию-подписку на сообщения. Это означает, что устройства подключаются к серверу MQTT и публикуют сообщения в топики (темы), а другие устройства могут подписаться на темы и получать сообщения.

MQTT предназначен для использования в системах, где требуется низкое потребление энергии, низкая латентность и высокая надежность. Он поддерживает различные уровни качества обслуживания (QoS), которые обеспечивают гарантированную доставку сообщений. QoS может быть установлен на уровне 0, 1 или 2. QoS 0 обеспечивает наиболее быструю доставку сообщений, но не гарантирует их доставку. QoS 1 гарантирует доставку сообщений хотя бы один раз, а QoS 2 гарантирует доставку сообщений ровно один раз.

MQTT поддерживает различные типы сообщений, такие как текстовые сообщения, бинарные данные, JSON-объекты и другие. Он также поддерживает различные форматы упаковки сообщений, такие как MQTT-SN для подключения устройств, не поддерживающих TCP/IP, и MQTT-S для работы в среде с ограниченными ресурсами.

MQTT широко используется в IoT-системах для связи между устройствами и серверами. Он позволяет создавать распределенные системы, в которых устройства могут обмениваться данными и взаимодействовать друг с другом. MQTT может быть использован для создания систем контроля и управления, систем мониторинга и диагностики, систем автоматизации и других.

В процессе работы с MQTT важно учитывать безопасность и надежность систем, в которых он используется. Необходимо использовать протоколы шифрования для защиты сообщений от несанкционированного доступа и модификации. Также важно обеспечить надежность связи между устройствами и серверами, чтобы гарантировать доставку сообщений в случае сбоев и отказов. MQTT поддерживает различные механизмы авторизации и аутентификации, которые позволяют контролировать доступ к топикам и сообщениям.

4. Описание Node-Red и его возможностей для обработки данных:
Node-Red - это интегрированная среда разработки, которая позволяет создавать визуальные схемы для обработки данных в реальном времени. Node-Red был разработан IBM и выпущен в 2013 году. Он основан на архитектуре клиент-сервер и поддерживает работу с различными

протоколами передачи данных, такими как MQTT, HTTP, WebSocket и другими. В Node-Red предусмотрено множество модулей (нод), которые позволяют выполнять различные операции с данными, такие как преобразование данных, фильтрация данных, агрегация данных и другие.

Node-Red позволяет создавать визуальные схемы для обработки данных путем соединения нод друг с другом. Ноды представляют собой функциональные блоки, которые могут выполнять различные операции, такие как получение данных из источника, преобразование данных, фильтрация данных, агрегация данных, отправка данных в приемник и другие. Ноды могут быть подключены друг к другу с помощью входных и выходных портов, которые позволяют передавать данные между нодами.

Node-Red позволяет создавать сложные схемы для обработки данных и интеграции с различными сервисами и устройствами. Например, можно создать схему для обработки данных с датчиков температуры, которая будет включать ноды для получения данных с датчиков, ноды для преобразования данных в формат CSV, ноды для отправки данных в базу данных, ноды для отображения данных на веб-странице и другие.

Node-Red поддерживает различные языки программирования, такие как JavaScript, Python, Node.js и другие. Он также поддерживает различные платформы, такие как Raspberry Pi, Linux, Windows, MacOS и другие. Node-Red может быть запущен локально на компьютере или в облаке, например, на сервисах Amazon Web Services, Microsoft Azure, Google Cloud Platform и других.

В процессе работы с Node-Red важно учитывать безопасность и надежность систем, в которых он используется. Необходимо использовать протоколы шифрования для защиты данных от несанкционированного доступа и модификации. Также важно обеспечить надежность связи между устройствами и серверами, чтобы гарантировать доставку данных в случае сбоев и отказов. Node-Red поддерживает различные механизмы авторизации и аутентификации, которые позволяют контролировать доступ к схемам и данным.

5. Введение в формат файлов .csv и его использование для хранения данных:
Формат .csv (Comma-Separated Values) - это текстовый формат файла, который используется для хранения данных в виде таблицы. Формат .csv был разработан для хранения данных в простых текстовых файлах, которые могут быть легко прочитаны и обработаны различными программами. В файле .csv каждая строка представляет собой запись в

таблице, а столбцы разделены запятыми или другими разделителями, такими как точка с запятой (;), двоеточие (:) или табуляция (\t).

Формат .csv поддерживается множеством программ, таких как Microsoft Excel, LibreOffice Calc, Google Sheets и другими. Он также поддерживается множеством языков программирования, таких как Python, Java, JavaScript и другими. Формат .csv используется для хранения данных измерений, статистических данных, результатов экспериментов, результатов исследований и других данных.

Формат .csv обладает рядом преимуществ, которые делают его популярным для хранения данных:

a.Простота: Формат .csv очень прост и понятен. Он состоит из текстовых строк, разделенных разделителями, что делает его легко читаемым и обрабатываемым.

b.Компактность: Формат .csv занимает мало места на диске, что делает его подходящим для хранения больших объемов данных.

c.Поддержка множества программ: Формат .csv поддерживается множеством программ для работы с таблицами, что делает его удобным для обмена данными между различными системами.

d.Универсальность: Формат .csv является стандартом де-факто для хранения данных в виде таблиц, что делает его удобным для обмена данными между различными системами.

В процессе работы с форматом .csv важно учитывать его ограничения и особенности. Формат .csv не поддерживает сложные структуры данных, такие как объекты и массивы. Также он не поддерживает форматирование данных, такие как форматирование даты и времени, форматирование чисел и другие. Важно выбирать подходящие разделители для столбцов, чтобы избежать ошибок при чтении файла. Также важно учитывать кодировку файла, чтобы избежать проблем с чтением файла в различных системах.

Гипотеза:

"Используя Raspberry Pi в качестве клиентского устройства и протокол MQTT для передачи данных, можно создать эффективную клиент-серверную платформу для измерения и мониторинга температуры в реальном времени, что позволит собирать статистику температуры и анализировать данные для оптимизации работы системы."

Эта гипотеза предполагает, что Raspberry Pi обладает достаточной производительностью и функциональностью для измерения температуры с помощью датчика DS18B20 и передачи данных с использованием протокола MQTT на сервер. Сервер, в свою очередь, может обрабатывать и сохранять данные в формате .csv для дальнейшего анализа и визуализации статистики температуры. Таким образом, созданная система будет способна обеспечивать точные и своевременные измерения температуры, а также позволит пользователям отслеживать и анализировать изменения температуры в реальном времени для оптимизации работы системы.

Алгоритм работы клиент-серверной платформы:

Алгоритм работы клиент-серверной платформы "Статистика температуры" включает следующие шаги:

1. Настройка Raspberry Pi:
   a. Установка операционной системы Raspberry Pi (например, Raspbian) на карту microSD.
   b. Подключение датчика температуры DS18B20 к Raspberry Pi.
   c. Установка необходимых пакетов и библиотек для работы с датчиком DS18B20 и протоколом MQTT.

2. Написание программы на C для измерения температуры:
   a. Импортирование необходимых библиотек.
   b. Настройка GPIO для работы с датчиком DS18B20.
   c. Получение адреса датчика DS18B20.
   d. Чтение значения температуры с датчика DS18B20.
   e. Запись значения температуры в файл .csv.

3. Настройка Raspberry Pi для передачи данных с использованием протокола MQTT:
   a. Установка библиотек MQTT.
   b. Создание издателя MQTT для передачи данных температуры.
   c. Настройка Raspberry Pi для подключения к брокеру MQTT.
   d. Настройка Raspberry Pi для публикации значения температуры в топик MQTT с указанными параметрами.

4. Настройка Node-Red для приёма данных из топика MQTT и сохранения данных в .csv файл на сервере:
   a. Установка Node-Red на сервер.
   b. Создание нового потока для обработки данных температуры.
   c. Добавление блока MQTT-input в поток для подключения к топику MQTT и приёма данных температуры.
   d. Добавление блока функция в поток для обработки данных температуры (например, преобразование данных в формат .csv).
   e. Добавление блока MQTT-output в поток для публикации данных температуры в другой топик MQTT (например, для отображения данных в веб-интерфейсе).
   f. Сохранение данных температуры в .csv файл на сервере.

5. Тестирование и отладка системы:
   a. Проверка работы датчика DS18B20 и программы на C для измерения температуры.
   b. Проверка работы Raspberry Pi для передачи данных температуры с

использованием протокола MQTT с помощью программы-подписчика.
c. Проверка работы Node-Red для приёма данных температуры от издателя MQTT и сохранения данных в .csv файл на сервере.
d. Отладка системы в случае обнаружения ошибок или несоответствий.

6. Описание результатов работы системы и анализ полученных данных:
   a. Описание работы системы "Статистика температуры" и ее основных функций.
   b. Анализ полученных данных о температуре (например, минимальная и максимальная температура, средняя температура за определенный период времени).
   c. Оценка эффективности системы и возможность дальнейшего улучшения ее функциональности.

Практическая часть:

1. Описание подключения датчика DS18B20 к Raspberry Pi: Датчик DS18B20 подключается к Raspberry Pi через интерфейс GPIO. Он требует всего одного провода данных для работы. Для подключения датчика DS18B20 к Raspberry Pi необходимо соединить вывод данных (DQ) датчика с выводом GPIO на Raspberry Pi. Также необходимо подключить общую землю (GND) между датчиком и Raspberry Pi. Для работы датчика DS18B20 необходимо также подключить питание (VDD) к источнику питания 3.3 В.

2. Для работы с датчиком DS18B20 на Raspberry Pi на языке C можно использовать библиотеку libgpiod, которая предоставляет низкоуровневый доступ к GPIO-пинам Raspberry Pi. Однако, для работы с датчиком DS18B20 на C можно использовать также библиотеку OneWire, которая предоставляет высокоуровневые функции для работы с датчиками на шине 1-Wire. Ниже приведен пример программы на C для измерения температуры с использованием датчика DS18B20 с библиотекой OneWire:

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <unistd.h>
5. #include <errno.h>
6. #include <time.h>
7. #include <wiringPi.h>
8. #include <wiringPiI2C.h>
9. #include <onewire.h>
10.#include <ds18b20.h>
11.
12.#define ONE_WIRE_BUS 4
13.
14.int main(void)
15.{
16.    int i;
17.    int err;
18.    int devCount;
19.    int temperature;
20.    double celsius;
21.
22.    // Initialize the library
23.    if (wiringPiSetup() == -1) {
24.        fprintf(stderr, "wiringPiSetup failed: %s\n", strerror(errno));
25.        return 1;
```

```
    }

    // Initialize the OneWire library
    if ((err = onewire_init(ONE_WIRE_BUS)) != 0) {
        fprintf(stderr, "onewire_init failed: %s\n", onewire_strerror(err));
        return 1;
    }

    // Get the number of devices on the OneWire bus
    if ((err = onewire_get_dev_count(&devCount)) != 0) {
        fprintf(stderr, "onewire_get_dev_count failed: %s\n",
    onewire_strerror(err));
        return 1;
    }

    // Loop through each device on the OneWire bus
    for (i = 0; i < devCount; i++) {
        // Search for the DS18B20 device
        if (ds18b20_search(i) != 0) {
            fprintf(stderr, "ds18b20_search failed: %s\n", ds18b20_strerror(err));
            return 1;
        }

        // Read the temperature from the DS18B20 device
        if (ds18b20_read_temp(&temperature) != 0) {
            fprintf(stderr, "ds18b20_read_temp failed: %s\n",
    ds18b20_strerror(err));
            return 1;
        }

        // Convert the temperature from integer to double
        celsius = (double)temperature / 1000.0;

        // Print the temperature in Celsius
        printf("Temperature: %.1f C\n", celsius);

        // Reset the search state
        ds18b20_reset_search();
    }

    return 0;
}
```

Эта программа на языке C использует библиотеки wiringPi и onewire для работы с GPIO-пинами и шиной 1-Wire соответственно. Она ищет все устройства на шине 1-Wire, ищет DS18B20-устройство, читает температуру с него и выводит результат на экран.

Комментарии к коду программы работы с датчиком DS18B20:

1. #include <stdio.h> - подключение заголовочного файла для работы с вводом-выводом.
2. #include <stdlib.h> - подключение заголовочного файла для работы со стандартными функциями.
3. #include <string.h> - подключение заголовочного файла для работы со строками.
4. #include <unistd.h> - подключение заголовочного файла для работы с системой.
5. #include <errno.h> - подключение заголовочного файла для работы с кодами ошибок.
6. #include <time.h> - подключение заголовочного файла для работы со временем.
7. #include <wiringPi.h> - подключение заголовочного файла для работы с GPIO на Raspberry Pi.
8. #include <wiringPiI2C.h> - подключение заголовочного файла для работы с I2C на Raspberry Pi.
9. #include <onewire.h> - подключение заголовочного файла для работы с однопроводной шиной на Raspberry Pi.
10. #include <ds18b20.h> - подключение заголовочного файла для работы с датчиком DS18B20.
11. #define ONE_WIRE_BUS 4 - определение номера пина GPIO для подключения датчика DS18B20 к Raspberry Pi.
12. int main(void) - начало функции main(), которая является точкой входа в программу.
13. int i; - объявление переменной для использования в цикле.

14. int err; - объявление переменной для хранения результатов операций с библиотеками.

15. int devCount; - объявление переменной для хранения количества устройств на однопроводной шине.

16. int temperature; - объявление переменной для хранения значения температуры.

17. double celsius; - объявление переменной для хранения значения температуры в градусах Цельсия.

18. // Initialize the library - инициализация библиотеки wiringPi для работы с GPIO на Raspberry Pi.

19. if (wiringPiSetup() == -1) - проверка результата инициализации библиотеки wiringPi.

20. {

21. fprintf(stderr, "wiringPiSetup failed: %s\n", strerror(errno)); - вывод сообщения об ошибке и кода ошибки.
22. return 1; - возврат ошибки при неудачной инициализации библиотеки wiringPi.

23. }

24. // Initialize the OneWire library - инициализация библиотеки onewire для работы с однопроводной шиной на Raspberry Pi.

25. if ((err = onewire_init(ONE_WIRE_BUS)) != 0) - проверка результата инициализации библиотеки onewire.

26. {

27. fprintf(stderr, "onewire_init failed: %s\n", onewire_strerror(err)); - вывод сообщения об ошибке и текста ошибки.
28. return 1; - возврат ошибки при неудачной инициализации библиотеки onewire.

29. }

30. // Get the number of devices on the OneWire bus - получение количества устройств на однопроводной шине.

31. if ((err = onewire_get_dev_count(&devCount)) != 0) - проверка результата получения количества устройств на однопроводной шине.

32. {

33. fprintf(stderr, "onewire_get_dev_count failed: %s\n", onewire_strerror(err)); - вывод сообщения об ошибке и текста ошибки.
34.
35. return 1; - возврат ошибки при неудачном получении количества устройств на однопроводной шине.

36. }

37. // Loop through each device on the OneWire bus - цикл для обработки каждого устройства на однопроводной шине.

38. for (i = 0; i < devCount; i++) - инициализация счетчика и условие для выхода из цикла.

39. {

40. // Search for the DS18B20 device - поиск датчика DS18B20 на однопроводной шине.
41. if (ds18b20_search(i) != 0) - проверка результата поиска датчика DS18B20.
42. {
43. fprintf(stderr, "ds18b20_search failed: %s\n", ds18b20_strerror(err)); - вывод сообщения об ошибке и текста ошибки.
44. return 1; - возврат ошибки при неудачном поиске датчика DS18B20.
45. }
46. // Read the temperature from the DS18B20 device - чтение значения температуры с датчика DS18B20.
47. if (ds18b20_read_temp(&temperature) != 0) - проверка результата чтения значения температуры с датчика DS18B20.
48. {
49. fprintf(stderr, "ds18b20_read_temp failed: %s\n", ds18b20_strerror(err)); - вывод сообщения об ошибке и текста ошибки.
50. return 1; - возврат ошибки при неудачном чтении значения температуры с датчика DS18B20.
51. }
52. // Convert the temperature from integer to double - преобразование значения температуры из целого числа в число с плавающей запятой.
53. celsius = (double)temperature / 1000.0; - преобразование значения температуры в градусы Цельсия.
54.
55. // Print the temperature in Celsius - вывод значения температуры в градусах Цельсия на экран.
56.

57. printf("Temperature: %.1f C\n", celsius); - вывод значения температуры в градусах Цельсия с одним знаком после запятой.
58. // Reset the search state - сброс состояния поиска устройств на однопроводной шине.
59. ds18b20_reset_search(); - сброс состояния поиска устройств на однопроводной шине.

60. }

61. return 0; - возврат успешного завершения программы.

Для компиляции и запуска этой программы необходимо установить библиотеки wiringPi и onewire. Для установки библиотек выполните следующие команды в терминале Raspberry Pi:

```
1. sudo apt-get update
2. sudo apt-get install wiringpi
3. sudo apt-get install libonewire-dev
```

После установки библиотек вы можете скомпилировать программу следующим образом:

```
4. gcc -o temperature temperature.c -lwiringPi -lonewire -lds18b20
```

Затем вы можете запустить программу следующим образом:

```
5. ./temperature
```

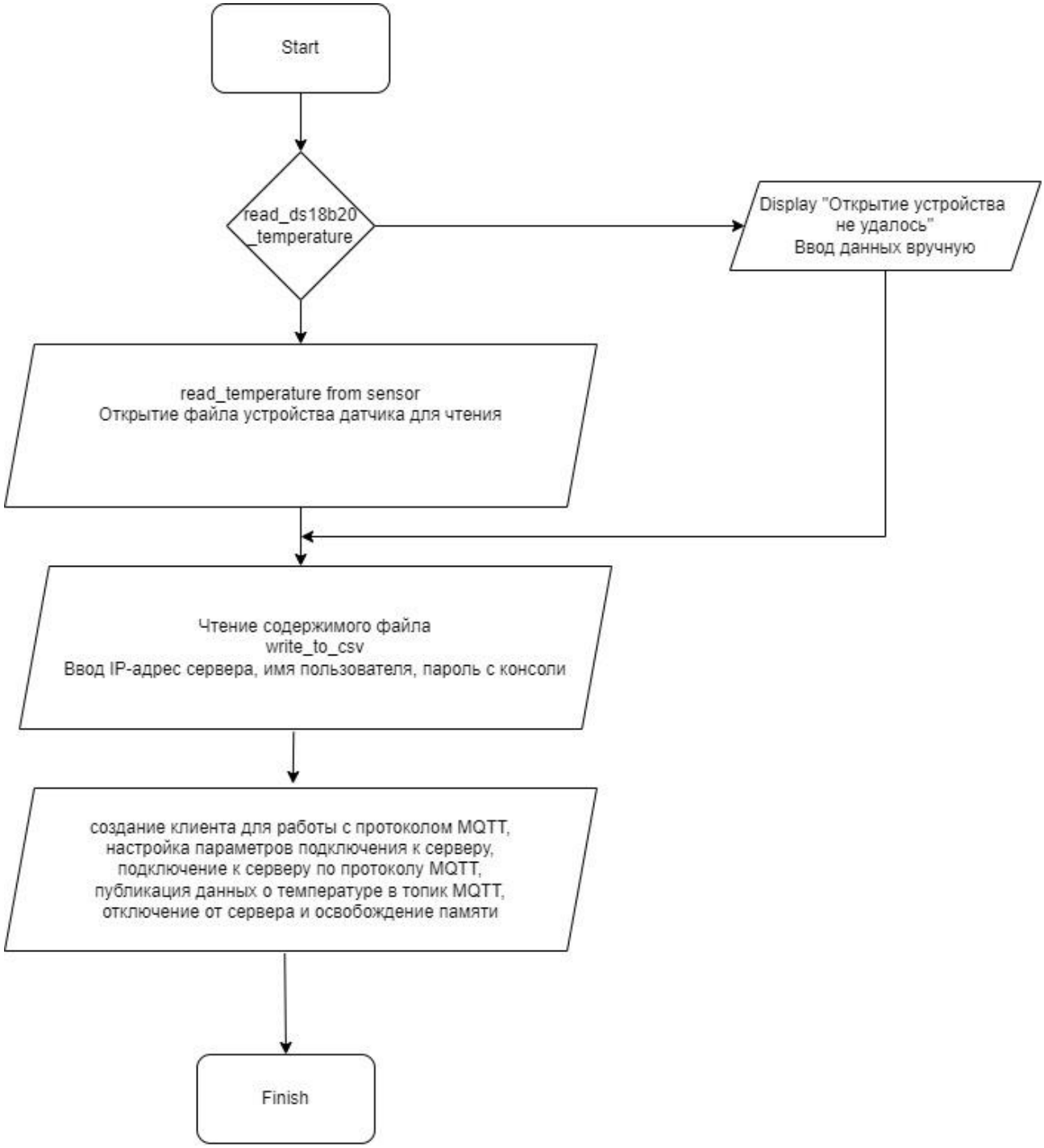
В результате вы увидите температуру в градусах Цельсия, измеренную с помощью датчика DS18B20.

3. Настройка Raspberry Pi для передачи данных с использованием протокола MQTT:

Для работы с протоколом MQTT на Raspberry Pi необходимо установить библиотеку paho-mqtt. Она позволяет легко публиковать и подписываться на топики MQTT. Для установки библиотек выполните следующие команды в терминале Raspberry Pi (файл commands):

```
1.    sudo apt-get install libssl-dev
2.
3.  git clone https://github.com/eclipse/paho.mqtt.c.git
4.  cd /путь до папки/paho.mqtt.c
5.  make
6.
7.  sudo make install
8.
9.  paho_c_sub -t my_topic --connection mqtt.eclipseprojects.io:1883
10.
11.paho_c_pub -t my_topic -m test --connection mqtt.eclipseprojects.io:1883
```

4. Блок-схема программы издателя:



5. Создание топика MQTT для передачи данных температуры: Для передачи значений температуры в MQTT broker необходимо создать топик MQTT. Топик MQTT представляет собой строку, которая определяет тему сообщения. Топик MQTT может быть произвольным и может включать несколько уровней. В нашем случае мы будем использовать топик " /node-red/temp". Пример файла для создания топика client_mqtt_file_new+sensor.c:

```
#include "stdio.h"
#include "stdint.h"
#include "time.h"
#include "stdlib.h"
#include "string.h"
#include <fcntl.h>
#include <unistd.h>

#include "MQTTClient.h"

#define CLIENTID    "ExampleClientPub"
#define TOPIC       "/node-red/temp"
#define PAYLOAD     "50"
#define QOS         1
#define TIMEOUT     10000L
double DELAY = 5;

struct sensor {
    uint16_t year;
    uint8_t month;
    uint16_t day;
    uint8_t hour;
    uint8_t minute;
    int8_t t;
};

void AddRecord (struct sensor* info, int number,
uint16_t year,uint8_t month,uint16_t day,uint8_t hour,uint8_t minute,int8_t t)
{
    info[number].year = year;
    info[number].month = month;
    info[number].day = day;
    info[number].hour = hour;
    info[number].minute = minute;
    info[number].t = t;
}
```

```
37.
38.void write_to_csv(int year, int month, int day, int hour, int minute, int
    temperature) {
39.    FILE *fp;
40.    char filename[] = "temp.csv";
41.
42.    fp = fopen(filename, "a"); // Открытие файла для добавления данных
43.    if (fp == NULL) {
44.        perror("Ошибка открытия файла");
45.        return;
46.    }
47.
48.    // Запись данных в файл в формате dddd;mm;dd;hh;mm;temperature
49.    fprintf(fp, "%04d;%02d;%02d;%02d;%02d;%d\n", year, month, day, hour,
    minute, temperature);
50.
51.    fclose(fp);
52.}
53.
54.int read_ds18b20_temperature() {
55.    const char *device_path = "/sys/bus/w1/devices/28-
    XXXXXXXXXXXX/w1_slave";// путь к файлу устройства DS18B20, 28-
    XXXXXXXXXXXX - идентификатор датчика
56.    char buf[256];
57.    char temp_str[6];
58.    int fd = open(device_path, O_RDONLY);
59.
60.    if(fd == -1) {
61.        perror("Открытие устройства не удалось");
62.        return -1;
63.    }
64.
65.    ssize_t num_read = read(fd, buf, sizeof(buf) - 1);
66.    if(num_read <= 0) {
67.        perror("Ошибка чтения");
68.        close(fd);
69.        return -1;
70.    }
71.
72.    buf[num_read] = '\0';
73.    char *temp_ptr = strstr(buf, "t=");
74.    if(temp_ptr == NULL) {
75.        printf("Ошибка: температура не найдена\n");
76.        close(fd);
```

```
77.         return -1;
78.     }
79.
80.     strncpy(temp_str, temp_ptr + 2, 5);
81.     temp_str[5] = '\0';
82.     int temperature = atoi(temp_str);
83.
84.     close(fd);
85.     return temperature / 1000; // Возвращаем температуру в градусах
    Цельсия
86. }
87.
88.
89. int main(int argc, char* argv[])
90. {
91.     int temperature = read_ds18b20_temperature(); // Температура,
    полученная от датчика
92.     // int temperature = 25; // Пример температуры, полученной от датчика
93.     time_t now = time(NULL);
94.     struct tm *now_tm = localtime(&now);
95.
96.     int year = now_tm->tm_year + 1900;
97.     int month = now_tm->tm_mon + 1;
98.     int day = now_tm->tm_mday;
99.     int hour = now_tm->tm_hour;
100.    int minute = now_tm->tm_min;
101.
102.    write_to_csv(year, month, day, hour, minute, temperature);
103.
104.    char address[50];
105.    char username[50];
106.    char password[50];
107.
108.    printf("Введите IP-адрес сервера: ");
109.    scanf("%49s", address);
110.    printf("Введите имя пользователя: ");
111.    scanf("%49s", username);
112.    printf("Введите пароль: ");
113.    scanf("%49s", password);
114.
115.    FILE *file;
116.    file = fopen("temp.csv","r");
117.
118.    MQTTClient client;
```

```
119.    MQTTClient_connectOptions conn_opts =
    MQTTClient_connectOptions_initializer;
120.    MQTTClient_message pubmsg = MQTTClient_message_initializer;
121.    MQTTClient_deliveryToken token;
122.    int rc;
123.
124.    MQTTClient_create(&client, address, CLIENTID,
125.        MQTTCLIENT_PERSISTENCE_NONE, NULL);
126.    conn_opts.keepAliveInterval = 20;
127.    conn_opts.cleansession = 1;
128.    conn_opts.username = username;
129.    conn_opts.password = password;
130.
131.
132.    if ((rc = MQTTClient_connect(client, &conn_opts)) !=
    MQTTCLIENT_SUCCESS)
133.    {
134.        printf("Failed to connect, return code %d\n", rc);
135.        exit(-1);
136.    }
137.
138.    struct sensor*info = malloc (365*24*60*sizeof(struct sensor));
139.
140.
141.
142.        int Y,M,D,H,Min,T;
143.        int r;
144.        int count=0;
145.
146.        for
    (;(r=fscanf(file,"%d;%d;%d;%d;%d;%d",&Y,&M,&D,&H,&Min,&T))>0;coun
    t++)
147.        {
148.            if (r<6)
149.            {
150.                char s[20], c;
151.                r = fscanf (file, "%[^\n]%c",s,&c);
152.                printf("Wrong format in line %s\n",s);
153.            }
154.            else
155.            {
156.                printf("%d %d %d %d %d %d\n",Y,M,D,H,Min,T);
157.                AddRecord(info,count,Y,M,D,H,Min,T);
158.            }
```

```
159.                         }
160.         fclose(file);
161.
162.         int i = 0;
163.
164.     while(1)
165.     {
166.
167.
168.         clock_t begin = clock();
169.
170.         char str[255];
171.         sprintf(str,"%d",info[i++].t);
172.         printf("%s,%d\n",str,i);
173.
174.         if(i>=count)
175.             i=0;
176.
177.
178.         pubmsg.payload = str;
179.         pubmsg.payloadlen = strlen(str);
180.         pubmsg.qos = QOS;
181.         pubmsg.retained = 0;
182.         MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
183.         //printf("Waiting for up to %d seconds for publication of %s\n"
184.         //        "on topic %s for client with ClientID: %s\n",
185.         //        (int)(TIMEOUT/1000), PAYLOAD, TOPIC, CLIENTID);
186.         rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
187.         //printf("Message with delivery token %d delivered\n", token);
188.
189.         while ((double)(clock() - begin)/CLOCKS_PER_SEC<DELAY)
190.         {}
191.     }
192.     MQTTClient_disconnect(client, 10000);
193.     MQTTClient_destroy(&client);
194.     return rc;
195. }
```

Комментарии к коду:

1. #include "stdio.h" - подключение стандартной библиотеки ввода-вывода.
2. #include "stdint.h" - подключение стандартной библиотеки для работы с типами данных.
3. #include "time.h" - подключение библиотеки для работы с временем.
4. #include "stdlib.h" - подключение стандартной библиотеки, предоставляющей функции для работы с памятью и преобразования типов данных.
5. #include "string.h" - подключение библиотеки для работы со строками.
6. #include <fcntl.h> - подключение библиотеки для работы с файловыми дескрипторами.
7. #include <unistd.h> - подключение библиотеки для работы с системными вызовами.
8. #include "MQTTClient.h" - подключение библиотеки для работы с протоколом MQTT.
9. #define CLIENTID "ExampleClientPub" - определение идентификатора клиента для MQTT.
10. #define TOPIC "/node-red/temp" - определение топика для публикации данных температуры.
11. #define PAYLOAD "50" - определение значения температуры для публикации.
12. #define QOS 1 - определение уровня качества обслуживания для публикации (QoS).
13. #define TIMEOUT 10000L - определение времени ожидания для публикации сообщения.
14. double DELAY = 5; - определение задержки между публикациями сообщений (в секундах).

18-25. struct sensor - определение структуры для хранения данных о температуре.

27-36. void AddRecord(struct sensor* info, int number, uint16_t year, uint8_t month, uint16_t day, uint8_t hour, uint8_t minute, int8_t t) - функция для добавления записи о температуре в структуру.

38-52. void write_to_csv(int year, int month, int day, int hour, int minute, int temperature) - функция для записи данных о температуре в файл .csv.

54-86. int read_ds18b20_temperature() - функция для чтения данных о температуре с датчика DS18B20.

90-195. int main(int argc, char* argv[]) - главная функция программы.

91. int temperature = read_ds18b20_temperature(); - чтение данных о температуре с датчика DS18B20.

96-100. time_t now = time(NULL); struct tm *now_tm = localtime(&now); - получение текущего времени и преобразование его в структуру struct tm.

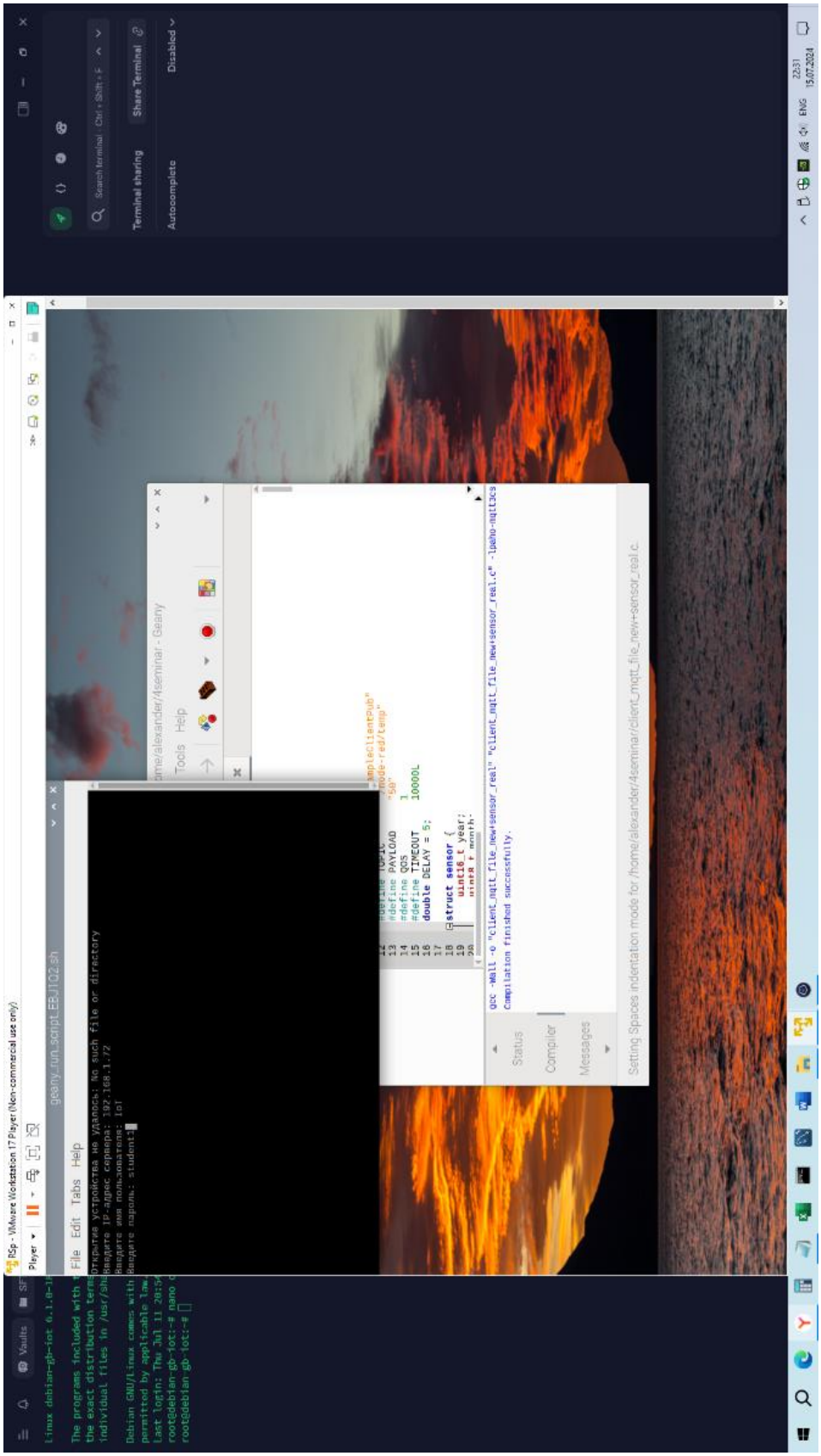
102. write_to_csv(year, month, day, hour, minute, temperature); - запись данных о температуре в файл .csv.

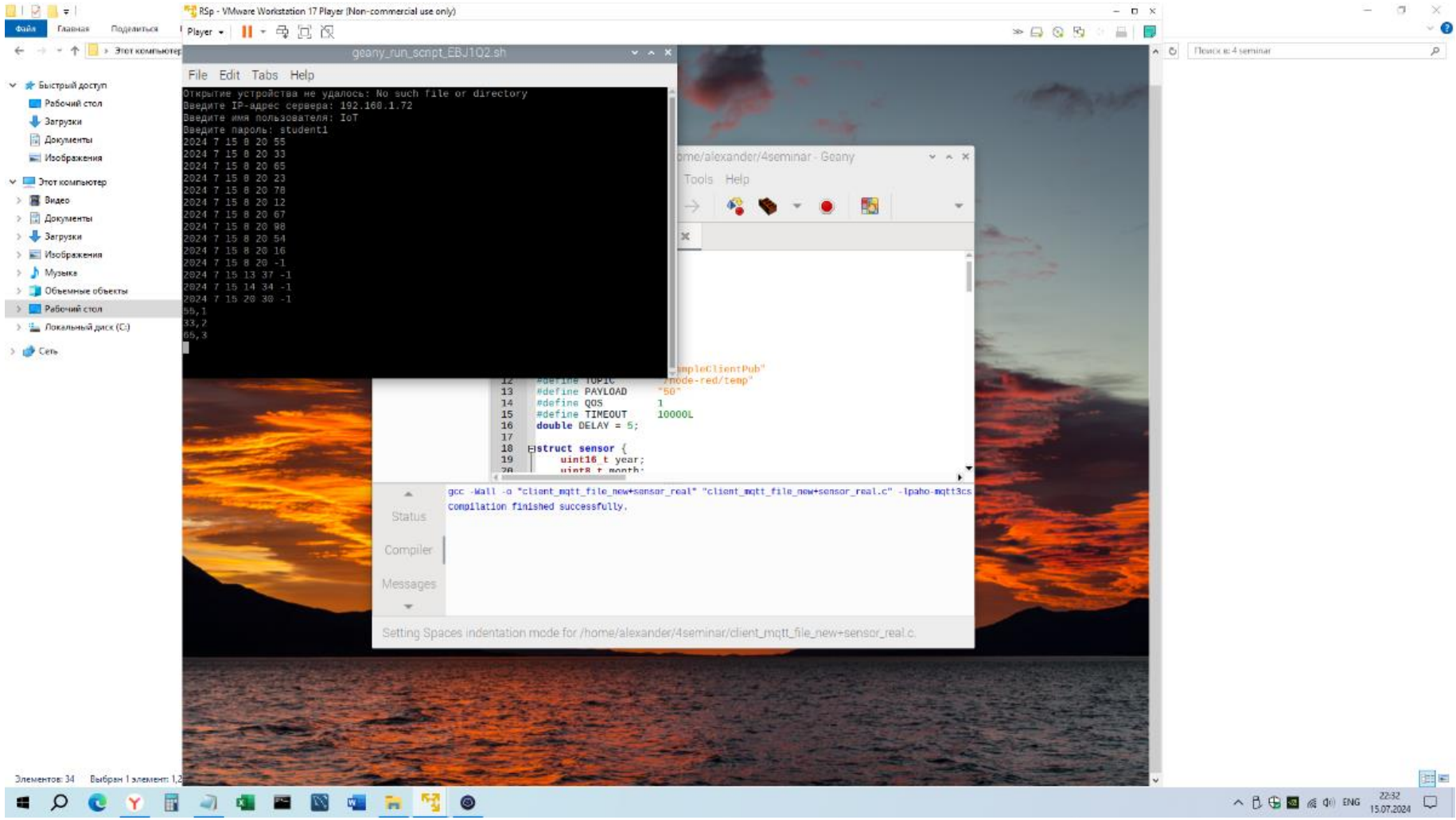
104-113. printf("Введите IP-адрес сервера: "); printf("Введите имя пользователя: "); printf("Введите пароль: "); - запрос у пользователя информации о сервере для подключения по протоколу MQTT.
115-120. MQTTClient client; MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer; MQTTClient_message pubmsg = MQTTClient_message_initializer; MQTTClient_deliveryToken token; - определение переменных для работы с протоколом MQTT.
124-125. MQTTClient_create(&client, address, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL); - создание клиента для работы с протоколом MQTT.
126-130. conn_opts.keepAliveInterval = 20; conn_opts.cleansession = 1; conn_opts.username = username; conn_opts.password = password; - настройка параметров подключения к серверу.
132-133. if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) - подключение к серверу по протоколу MQTT.

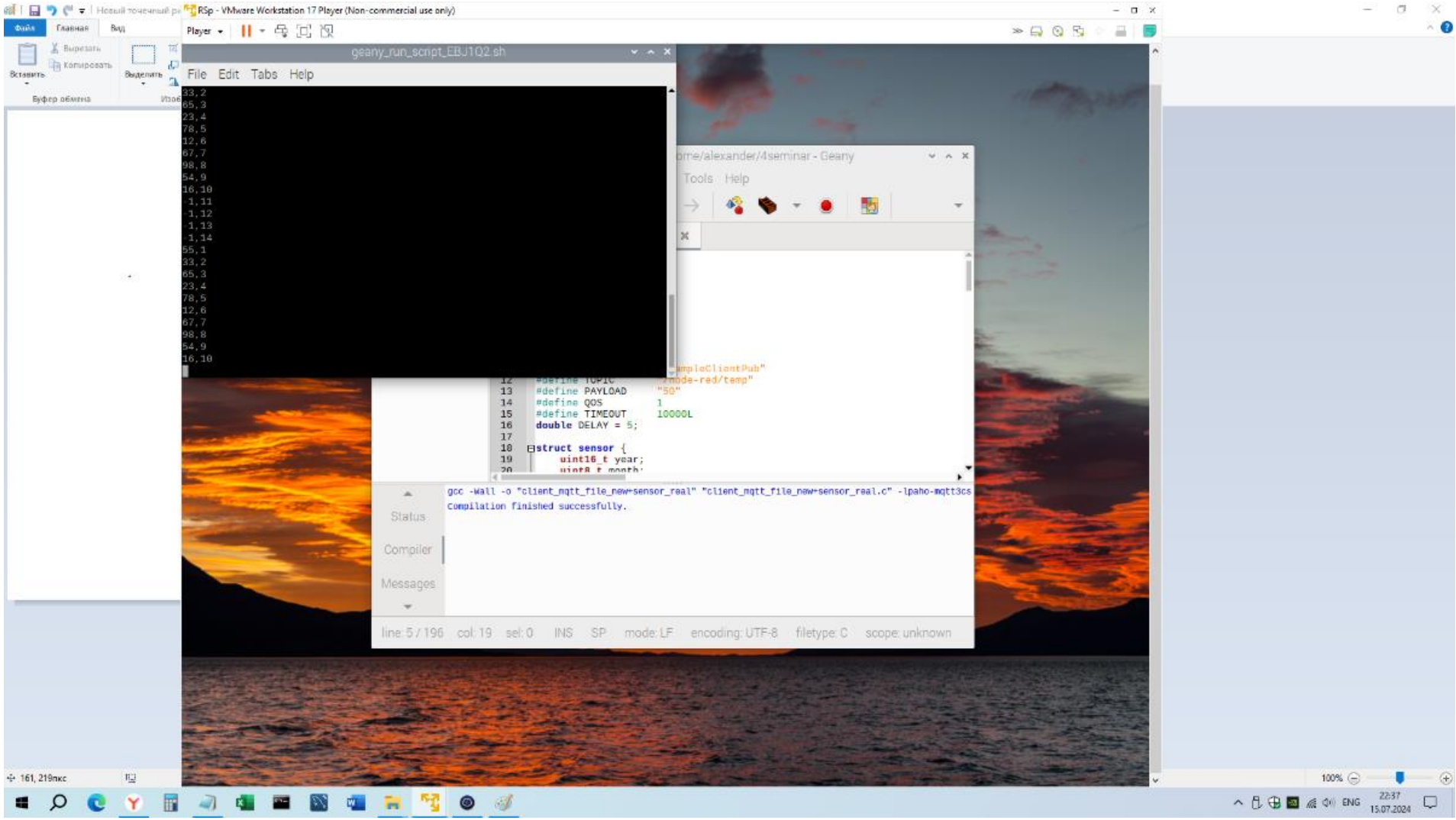
138. struct sensor* info = malloc (365*24*60*sizeof(struct sensor)); - выделение памяти для хранения данных о температуре.

141-160. int i = 0; while(1) - цикл для публикации данных о температуре в топик MQTT.
170-180. char str[255]; sprintf(str,"%d",info[i++].t); pubmsg.payload = str; pubmsg.payloadlen = strlen(str); pubmsg.qos = QOS; pubmsg.retained = 0; MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token); - формирование и публикация сообщения о температуре в топик MQTT.
182-184. rc = MQTTClient_waitForCompletion(client, token, TIMEOUT); - ожидание завершения публикации сообщения.
189-190. while ((double)(clock() - begin)/CLOCKS_PER_SEC<DELAY) {} - ожидание заданной задержки между публикациями сообщений.
192-193. MQTTClient_disconnect(client,
10000); MQTTClient_destroy(&client); - отключение от сервера и освобождение памяти.
194-195. return rc; - возврат кода ошибки или успеха.

Демонстрация работы издателя при передаче информации о температуре:




RSp - VMware Workstation 17 Player (Non-commercial use only)
geany_run_script_EBJ1Q2.sh
File Edit Tabs Help
Открытие устройства не удалось: No such file or directory
Введите IP-адрес сервера: 192.168.1.72
Введите имя пользователя: IoT
Введите пароль: student1
2024 7 15 8 20 55
2024 7 15 8 20 33
2024 7 15 8 20 65
2024 7 15 8 20 23
2024 7 15 8 20 78
2024 7 15 8 20 12
2024 7 15 8 20 67
2024 7 15 8 20 98
2024 7 15 8 20 54
2024 7 15 8 20 16
2024 7 15 8 20 -1
2024 7 15 13 37 -1
2024 7 15 14 34 -1
2024 7 15 20 38 -1
55,1
33,2
65,3
Tools Help
#define PAYLOAD "50"
#define QOS 1
#define TIMEOUT 10000L
double DELAY = 5;
struct sensor {
uint16_t year;
gcc -Wall -o "client_mqtt_file_new+sensor_real" "client_mqtt_file_new+sensor_real.c" -lpaho-mqtt3cs
compilation finished successfully.
Status
Compiler
Messages
Setting Spaces indentation mode for /home/alexander/4seminar/client_mqtt_file_new+sensor_real.c.
ENG
22:32
15.07.2024


RSp - VMware Workstation 17 Player (Non-commercial use only)
Player
geany_run_script_EBJ1Q2.sh
File Edit Tabs Help
33,2
65,3
23,4
78,5
12,6
67,7
98,8
54,9
16,10
-1,11
-1,12
-1,13
-1,14
55,1
33,2
65,3
23,4
78,5
12,6
67,7
98,8
54,9
16,10
Tools Help
#define PAYLOAD "50"
#define QOS 1
#define TIMEOUT 10000L
double DELAY = 5;
struct sensor {
uint16_t year;
gcc -Wall -o "client_mqtt_file_new+sensor_real" "client_mqtt_file_new+sensor_real.c" -lpaho-mqtt3cs
Compilation finished successfully.
Status
Compiler
Messages
line: 5 / 196 col: 19 sel: 0 INS SP mode: LF encoding: UTF-8 filetype: C scope: unknown
ENG
15.07.2024

6. Настройка Node-Red для приёма данных из топика MQTT и сохранения данных в .csv файл на сервере:
   Для работы с MQTT broker в Node-Red необходимо установить модуль "node-red-contrib-mqtt". Он позволяет подключаться к MQTT broker и получать данные из топиков MQTT. Для сохранения данных в .csv файл на сервере необходимо использовать модуль "csv". Ниже приведен пример кода на Node-Red для приёма данных из топика MQTT и сохранения их в .csv файл на сервере:

```
[
    {
        "id": "702656a.59f1a",
        "type": "mqtt in",
        "z": "6058c0e.47a8",
        "name": "",
        "topic": "/node-red/temp",
        "qos": "2",
        "broker": "989e620.4d236",
        "x": 210,
        "y": 160,
        "wires": [
            ["e62f97f.a88e9"]
        ]
    },
    {
        "id": "36c2f97f.a88e9",
        "type": "function",
        "z": "6058c0e.47a8",
        "name": "Parse temperature",
        "func": "msg.payload = parseFloat(msg.payload);\nreturn msg;",
        "outputs": 1,
        "noerr": 0,
        "x": 400,
        "y": 160,
        "wires": [
            ["1836f31.9e239"]
        ]
    },
    {
        "id": "1836f31.9e239",
        "type": "csv",
        "z": "6058c0e.47a8",
        "name": "",
        "filename": "/home/pi/temperature.csv",
```

```
    "appendNewline": true,
    "append": false,
    "separator": ",",
    "columns": [
      "temperature"
    ],
    "x": 600,
    "y": 160,
    "wires": []
  },
  {
    "id": "989e620.4d236",
    "type": "mqtt-broker",
    "z": "",
    "name": "MQTT broker",
    "broker": "localhost",
    "port": "1883",
    "clientid": "",
    "usetls": false,
    "compatmode": true,
    "keepalive": "60",
    "cleansession": true,
    "birthTopic": "",
    "birthQos": "0",
    "birthPayload": "",
    "closeTopic": "",
    "closeQos": "0",
    "closePayload": "",
    "willTopic": "",
    "willQos": "0",
    "willPayload": ""
  }
]
```

Комментарии к настройкам Node-Red:

1. "id": "702656a.59f1a" - уникальный идентификатор узла.
2. "type": "mqtt in" - тип узла, в данном случае - узел для получения сообщений по протоколу MQTT.
3. "z": "6058c0e.47a8" - идентификатор области, к которой принадлежит узел.
4. "name": "" - имя узла (необязательно).
5. "topic": "/node-red/temp" - топик для подписки на сообщения MQTT.

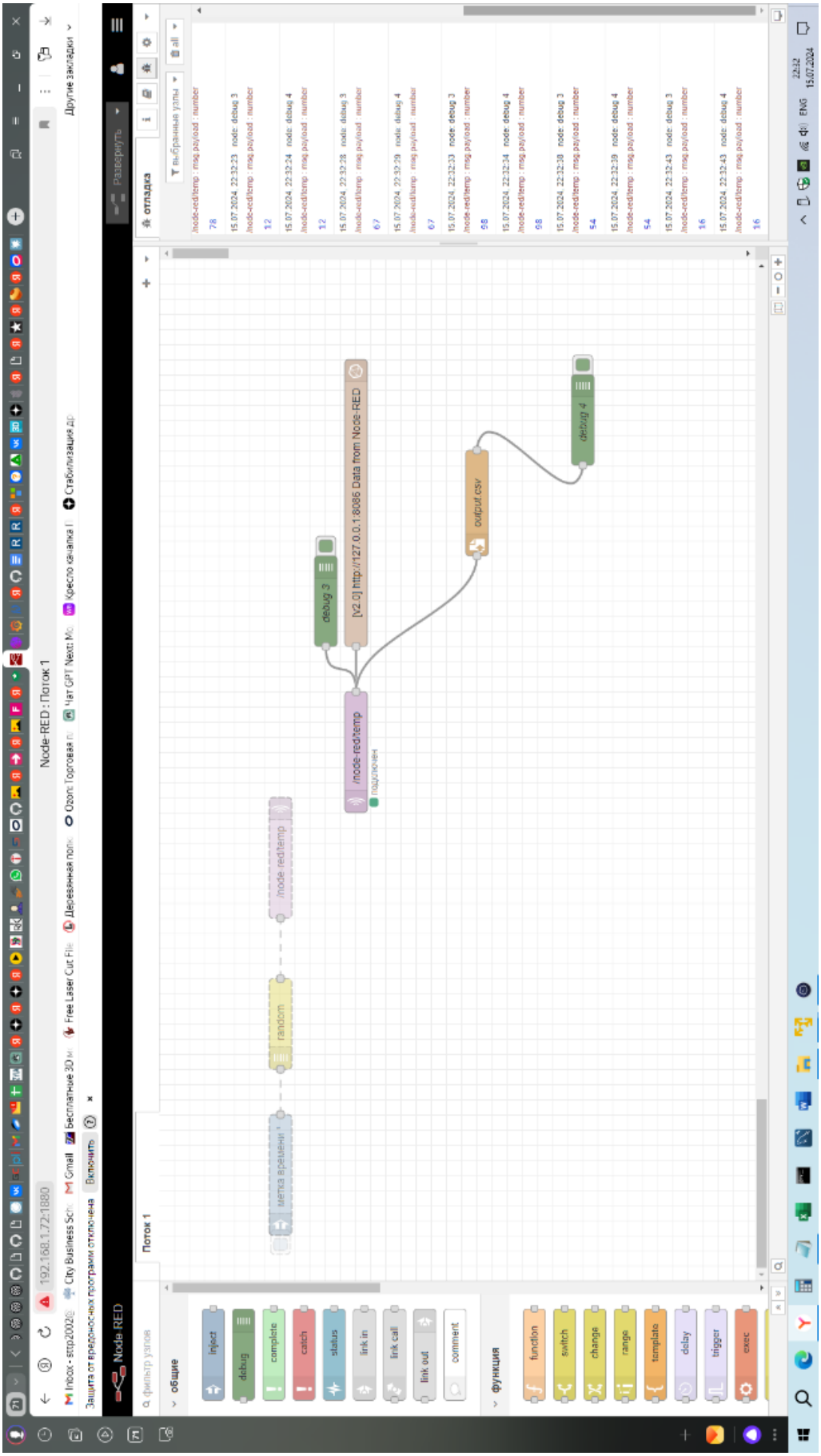
6. "qos": "2" - уровень качества обслуживания (QoS) для получаемых сообщений MQTT.
7. "broker": "989e620.4d236" - идентификатор узла брокера MQTT, с которым узел будет работать.
8. "x": 210, "y": 160 - координаты узла на диаграмме.
9. "wires": [["e62f97f.a88e9"]] - массив связей с другими узлами.
10. "id": "36c2f97f.a88e9" - уникальный идентификатор узла.
11. "type": "function" - тип узла, в данном случае - узел для выполнения функции.
12. "name": "Parse temperature" - имя узла (необязательно).
13. "func": "msg.payload = parseFloat(msg.payload);\nreturn msg;" - функция, которая будет выполняться узлом. В данном случае, она преобразует полученное значение температуры в число с плавающей запятой.
14. "outputs": 1 - количество выходных соединений узла.
15. "noerr": 0 - флаг, указывающий, что узел должен продолжать выполнение даже в случае ошибки.
16. "x": 400, "y": 160 - координаты узла на диаграмме.
17. "wires": [["1836f31.9e239"]] - массив связей с другими узлами.
18. "id": "1836f31.9e239" - уникальный идентификатор узла.
19. "type": "csv" - тип узла, в данном случае - узел для записи данных в файл .csv.
20. "name": "" - имя узла (необязательно).
21. "filename": "/home/pi/temperature.csv" - путь к файлу .csv для записи данных.
22. "appendNewline": true - флаг, указывающий, что необходимо добавлять новую строку после каждой записи в файл.
23. "append": false - флаг, указывающий, что необходимо добавлять новые данные в конец файла (если значение равно true).
24. "separator": "," - разделитель между полями в файле .csv.
25. "columns": ["temperature"] - массив названий столбцов в файле .csv.
26. "x": 600, "y": 160 - координаты узла на диаграмме.
27. "wires": [] - массив связей с другими узлами.
28. "id": "989e620.4d236" - уникальный идентификатор узла.
29. "type": "mqtt-broker" - тип узла, в данном случае - узел для работы с брокером MQTT.
30. "name": "MQTT broker" - имя узла (необязательно).
31. "broker": "localhost" - адрес брокера MQTT.
32. "port": "1883" - порт для подключения к брокеру MQTT.
33. "clientid": "" - идентификатор клиента MQTT (необязательно).
34. "usetls": false - флаг, указывающий, что необходимо использовать защищенное соединение (TLS) с брокером MQTT (если значение равно true).
35. "compatmode": true - флаг, указывающий, что необходимо использовать режим совместимости с MQTT v3.1 (если значение равно true).

36. "keepalive": "60" - интервал времени (в секундах) для отправки сообщений о поддержании соединения с брокером MQTT.
37. "cleansession": true - флаг, указывающий, что необходимо очищать сессию после отключения клиента от брокера MQTT (если значение равно true).

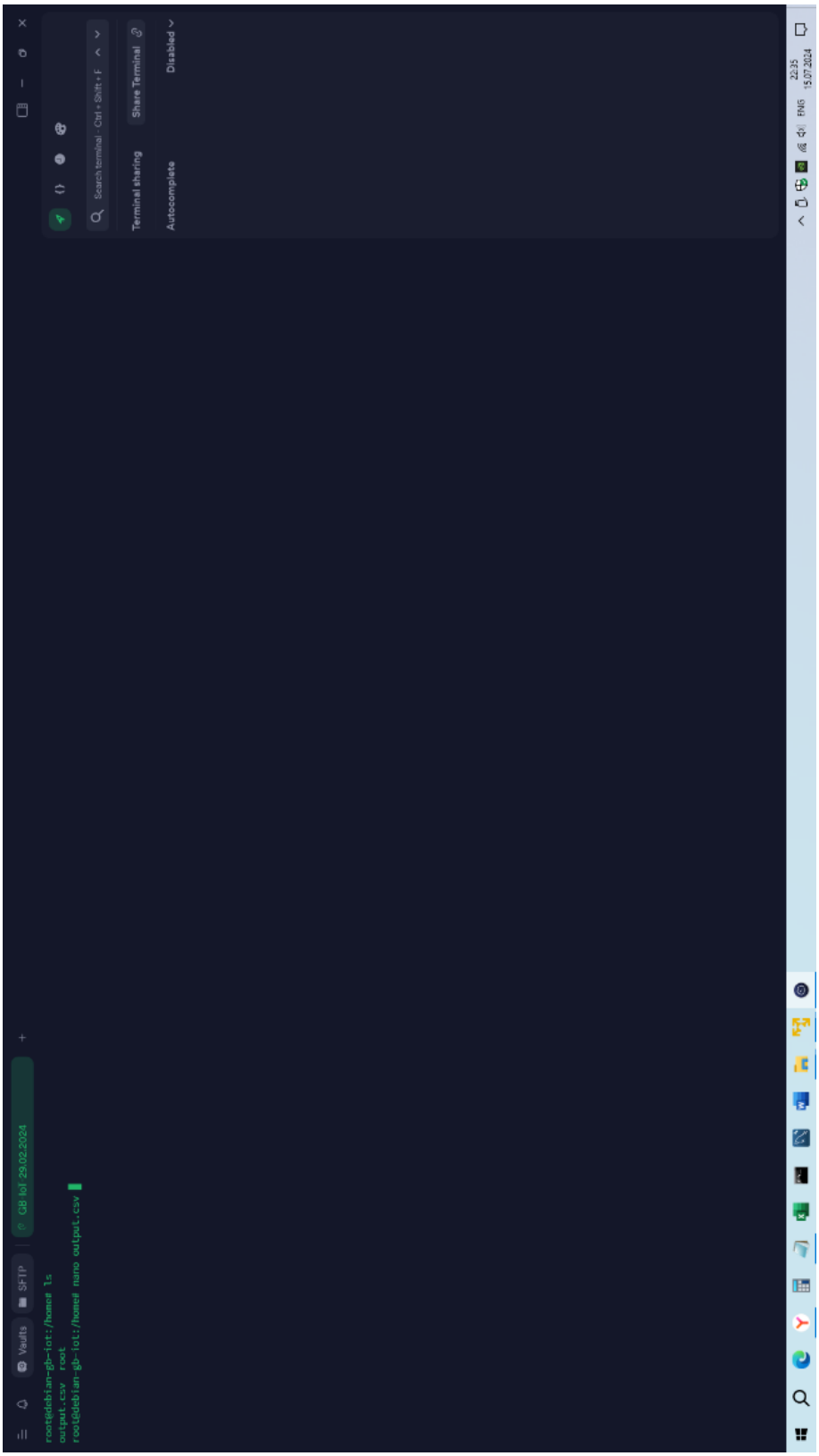
38. Остальные параметры - дополнительные настройки для работы с брокером MQTT.

39. "x": 210, "y": 160 - координаты узла на диаграмме.
40. "wires": [] - массив связей с другими узлами.

Демонстрация работы Node-Red на приеме информации от издателя:



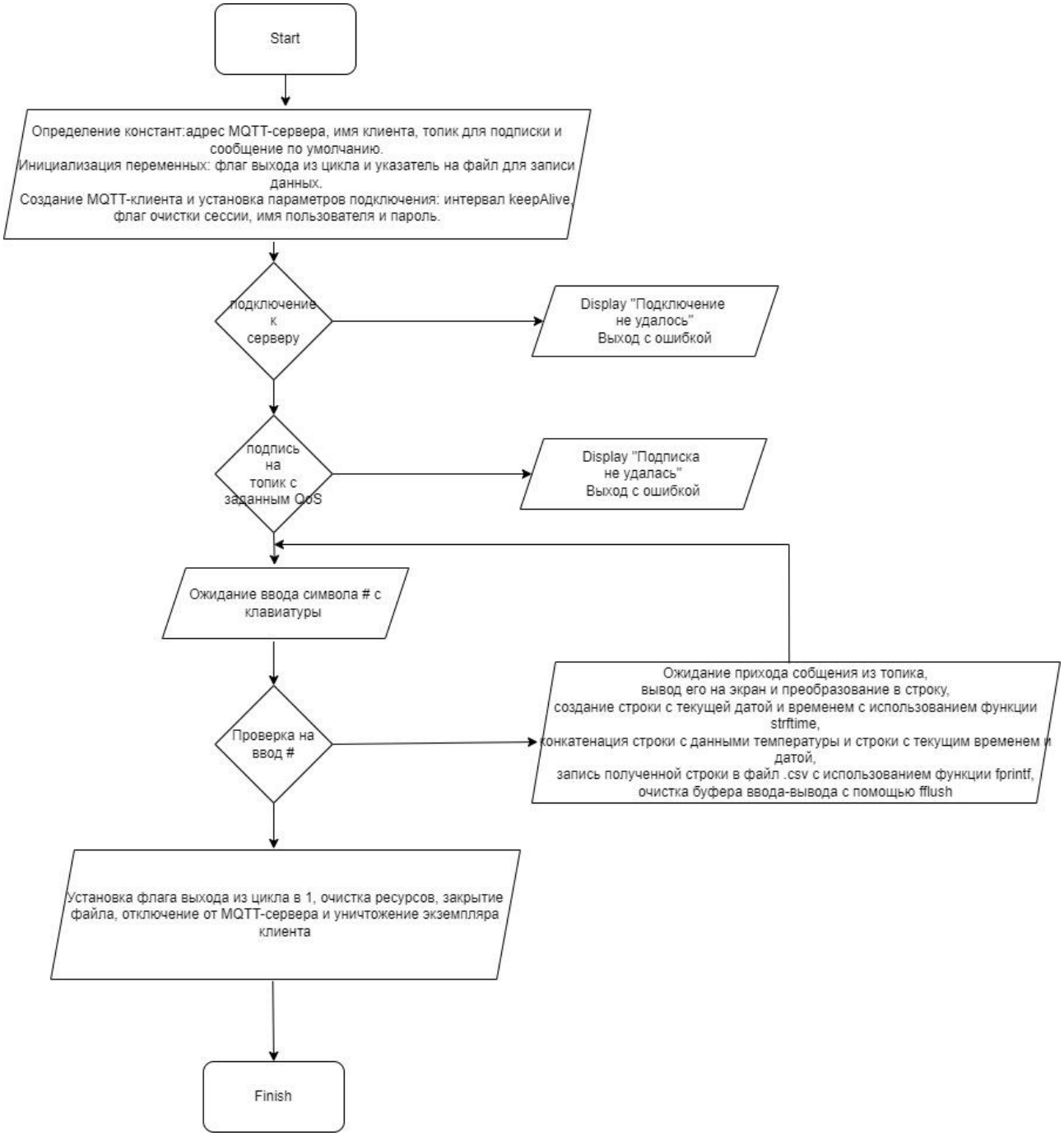
Демонстрация записи данных в .csv файл на сервере:




Vaults
SFTP
GB-IoT-29.02.2024
GNU nano 7.2
output.csv
Search terminal - Ctrl + Shift + F
Terminal sharing
Share Terminal
Autooomplete
Disabled
[Read 5925 lines]
^G Help
^X Exit
^O Write Out
^R Read File
^W Where Is
^\ Replace
^K Cut
^U Paste
^T Execute
^J Justify
^C Location
^/ Go To Line
M-U Undo
M-E Redo
M-A Set Mark
M-6 Copy
M-] To Bracket
M-Q Where Was
M-(Previous
M-) Next
M-B Back
M-F Forward
ENG
22:36
15.07.2024

7. Тестирование и отладка системы:
   Перед запуском системы необходимо проверить правильность подключения датчика DS18B20 к Raspberry Pi, правильность настройки MQTT broker и правильность настройки Node-Red. Для этого можно использовать тестовые функции в программе на C и в схеме на Node-Red.

8. Блок-схема программы подписчика:



Пример программы для проверки работы подписчика sub_mqtt_file.c:

```
#include "MQTTClient.h"
#include "MQTTClientPersistence.h"
#include "pubsub_opts.h"

#include <stdio.h>
#include <stdbool.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include "time.h"

#define ADDRESS "192.168.1.72:1883"
#define CLIENTID "ExampleClientPub"
#define TOPIC "/node-red/temp"
#define PAYLOAD "Hello World!"
#define QOS 1
#define TIMEOUT 10000L


int main()
{
  bool isExit = 0;

  FILE *file;
  file = fopen("temp1.csv","w");

  MQTTClient client;
  MQTTClient_connectOptions conn_opts =
MQTTClient_connectOptions_initializer;

  int rc;

  MQTTClient_create(&client, ADDRESS, CLIENTID,
            MQTTCLIENT_PERSISTENCE_NONE, NULL);
  conn_opts.keepAliveInterval = 20;
  conn_opts.cleansession = 1;
  conn_opts.username = "IoT";
  conn_opts.password = "student1";

  if ((rc = MQTTClient_connect(client, &conn_opts)) !=
MQTTCLIENT_SUCCESS)
  {
```

```
41.             printf("Failed to connect, return code %d\n", rc);
42.             exit(-1);
43.  }
44.  printf("Success in connect, return code %d\n", rc);
45.  rc = MQTTClient_subscribe(client, TOPIC, QOS);
46.  if (rc != MQTTCLIENT_SUCCESS && rc !=QOS)
47.  {
48.             fprintf(stderr, "Error %d subscribing to topic %s\n", rc, TOPIC);
49.             exit(-2);
50.  }
51.  printf("Success %d subscribing to topic %s\n", rc, TOPIC);
52.  printf("Waiting for the message\n");
53.  while(isExit!=1)
54.  {
55.             time_t mytime = time (NULL);
56.             struct tm *now = localtime(&mytime);
57.
58.             char* topicName = TOPIC;
59.             int topicLen = strlen(PAYLOAD);
60.             MQTTClient_message* message = NULL;
61.
62.             rc = MQTTClient_receive(client, &topicName, &topicLen,
    &message, 1000);
63.             if (rc == MQTTCLIENT_DISCONNECTED)
64.                       rc = MQTTClient_connect(client, &conn_opts);
65.             else if (message)
66.             {
67.                       printf("%s\n", (char*)message->payload);
68.
69.                       char str[255]={0};
70.                       char time[20];
71.                       char date[20];
72.
73.                       strftime(date,sizeof(str),"%D",now);
74.                       strftime(time,sizeof(str),"%T",now);
75.
76.                       strcat(str,(char*)message->payload);
77.                       printf("%s\n",str);
78.                       printf("%s\n",time);
79.                       printf("%s\n",date);
80.                       fprintf(file, "Temp: %s Time: %s Date: %s\n", str,
    time,date);
81.
82.                       fflush(stdout);
```

```
83.                     MQTTClient_freeMessage(&message);
84.                     MQTTClient_free(topicName);
85.                     printf("\n=> Press any key and Enter, # to stop
    connection\n");
86.                     char c;
87.                     while((c=getchar())!='\n')
88.                             if(c=='#')
89.                             {
90.                                     isExit=1;
91.                             }
92.             }
93.   }
94.exit:
95.   fclose(file);
96.   MQTTClient_disconnect(client, 10000);
97.   MQTTClient_destroy(&client);
98.   return EXIT_SUCCESS;
99.}
```

Комментарии к коду для проверки работы подписчика:

1. #include "MQTTClient.h" - подключение заголовочного файла для работы с библиотекой MQTT.
2. #include "MQTTClientPersistence.h" - подключение заголовочного файла для работы с персистентностью MQTT.
3. #include "pubsub_opts.h" - подключение заголовочного файла для работы с опциями публикации и подписки.
4. #include <stdio.h> - подключение заголовочного файла для работы с вводом-выводом.
5. #include <stdbool.h> - подключение заголовочного файла для работы со логическими значениями.
6. #include <signal.h> - подключение заголовочного файла для работы с сигналами.
7. #include <string.h> - подключение заголовочного файла для работы со строками.
8. #include <stdlib.h> - подключение заголовочного файла для работы со стандартными функциями.
9. #include "time.h" - подключение заголовочного файла для работы со временем.
10. #define ADDRESS "192.168.1.72:1883" - определение адреса брокера MQTT.

11. #define CLIENTID "ExampleClientPub" - определение идентификатора клиента MQTT.
12. #define TOPIC "/node-red/temp" - определение топика для публикации и подписки.
13. #define PAYLOAD "Hello World!" - определение текста сообщения для публикации.
14. #define QOS 1 - определение уровня качества обслуживания (QoS) для публикации сообщения.
15. #define TIMEOUT 10000L - определение времени ожидания операций с брокером MQTT.
16. int main() - начало функции main().
17. {
18. bool isExit = 0; - объявление и инициализация переменной для управления выходом из программы.
19. FILE *file; - объявление указателя на файл для записи данных в файл temp1.csv.
20. file = fopen("temp1.csv","w"); - открытие файла для записи.
21. MQTTClient client; - объявление переменной для работы с клиентом MQTT.
22. MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer; - объявление и инициализация структуры для настройки соединения с брокером MQTT.
23. int rc; - объявление переменной для хранения результатов операций с брокером MQTT.
24. MQTTClient_create(&client, ADDRESS, CLIENTID, - создание клиента MQTT с указанными параметрами.
25. MQTTCLIENT_PERSISTENCE_NONE, NULL);
26. conn_opts.keepAliveInterval = 20; - настройка интервала ожидания сообщений о поддержании соединения с брокером MQTT.
27. conn_opts.cleansession = 1; - настройка использования чистой сессии для соединения с брокером MQTT.
28. conn_opts.username = "IoT"; - настройка имени пользователя для соединения с брокером MQTT.
29. conn_opts.password = "student1"; - настройка пароля для соединения с брокером MQTT.
30. if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) - попытка установки соединения с брокером MQTT с использованием указанных параметров.
31. {
32. printf("Failed to connect, return code %d\n", rc); - вывод сообщения об ошибке подключения к брокеру MQTT.
33. exit(-1); - завершение программы с ошибкой.

34. }
35. printf("Success in connect, return code %d\n", rc); - вывод сообщения о успешном подключении к брокеру MQTT.
36. rc = MQTTClient_subscribe(client, TOPIC, QOS); - подписка на топик с указанными параметрами.
37. if (rc != MQTTCLIENT_SUCCESS && rc !=QOS) - проверка результата подписки на топик.
38. {
39. fprintf(stderr, "Error %d subscribing to topic %s\n", rc, TOPIC); - вывод сообщения об ошибке подписки на топик.
40. exit(-2); - завершение программы с ошибкой.
41. }
42. printf("Success %d subscribing to topic %s\n", rc, TOPIC); - вывод сообщения о успешной подписке на топик.
43. printf("Waiting for the message\n"); - сообщение о том, что программа ждет сообщения на подписанном топике.
44. while(isExit!=1) - цикл ожидания сообщений на подписанном топике.
45. {
46. time_t mytime = time (NULL); - получение текущего времени.
47. struct tm *now = localtime(&mytime); - преобразование текущего времени в локальное время.
48. char* topicName = TOPIC; - указатель на топик, на который получено сообщение.
49. int topicLen = strlen(PAYLOAD); - длина текста сообщения для публикации.
50. MQTTClient_message* message = NULL; - указатель на сообщение, полученное на подписанном топике.
51. rc = MQTTClient_receive(client, &topicName, &topicLen, &message, 1000); - получение сообщения на подписанном топике с указанным временем ожидания.
52. if (rc == MQTTCLIENT_DISCONNECTED) - проверка состояния подключения к брокеру MQTT.
53. rc = MQTTClient_connect(client, &conn_opts); - повторная попытка установки соединения с брокером MQTT.
54. else if (message) - проверка получения сообщения на подписанном топике.
55. {
56. printf("%s\n", (char*)message->payload); - вывод текста сообщения.
57. char str[255]={0}; - объявление массива для хранения текста сообщения.
58. char time[20]; - объявление массива для хранения времени сообщения.
59. char date[20]; - объявление массива для хранения даты сообщения.
60. strftime(date,sizeof(str),"%D",now); - преобразование текущей даты в строку формата "%D".
61. strftime(time,sizeof(str),"%T",now); - преобразование текущего времени в строку формата "%T".

62. strcat(str,(char*)message->payload); - добавление текста сообщения в массив str.
63. printf("%s\n",str); - вывод текста сообщения и строки, полученной из массива str.
64. printf("%s\n",time); - вывод текущего времени.
65. printf("%s\n",date); - вывод текущей даты.
66. fprintf(file, "Temp: %s Time: %s Date: %s\n", str, time,date); - запись строки с данными о сообщении в файл temp1.csv.
67. fflush(stdout); - выполнение выходного буфера для вывода сообщений на экран.
68. MQTTClient_freeMessage(&message); - освобождение памяти, выделенной под сообщение.
69. MQTTClient_free(topicName); - освобождение памяти, выделенной под топик.
70. printf("\n=> Press any key and Enter, # to stop connection\n"); - сообщение о том, что для остановки подключения необходимо нажать любую клавишу и Enter, а для выхода из программы - символ #.
71. char c; - объявление переменной для хранения символа, введенного с клавиатуры.
72. while((c=getchar())!='\n') - цикл ожидания ввода символа с клавиатуры.
73. if(c=='#') - проверка введенного символа.
74. {
75. isExit=1; - установка флага для выхода из цикла ожидания сообщений на подписанном топике.
76. }
77. }
78. }
79. exit: - метка для выхода из цикла ожидания сообщений на подписанном топике.
80. fclose(file); - закрытие файла temp1.csv.
81. MQTTClient_disconnect(client, 10000); - разрыв соединения с брокером MQTT с указанным временем ожидания.
82. MQTTClient_destroy(&client); - освобождение памяти, выделенной под клиента MQTT.
83. return EXIT_SUCCESS; - завершение программы с кодом успешного завершения.
84. }

Заголовочный файл для работы подписчика pubsub_opts.h:

```
#if !defined(PUBSUB_OPTS_H)
#define PUBSUB_OPTS_H

#include "MQTTAsync.h"
#include "MQTTClientPersistence.h"

struct pubsub_opts
{
  /* debug app options */
  int publisher;  /* publisher app? */
  int quiet;
  int verbose;
  int tracelevel;
  char* delimiter;
  int maxdatalen;
  /* message options */
  char* message;
  char* filename;
  int stdin_lines;
  int stdlin_complete;
  int null_message;
  /* MQTT options */
  int MQTTVersion;
  char* topic;
  char* clientid;
  int qos;
  int retained;
  char* username;
  char* password;
  char* host;
  char* port;
  char* connection;
  int keepalive;
  /* will options */
  char* will_topic;
  char* will_payload;
  int will_qos;
  int will_retain;
  /* TLS options */
  int insecure;
  char* capath;
  char* cert;
```

```
43.   char* cafile;
44.   char* key;
45.   char* keypass;
46.   char* ciphers;
47.   char* psk_identity;
48.   char* psk;
49.   /* MQTT V5 options */
50.   int message_expiry;
51.   struct {
52.               char *name;
53.               char *value;
54.   } user_property;
55.   /* HTTP proxies */
56.   char* http_proxy;
57.   char* https_proxy;
58.};
59.
60.typedef struct
61.{
62.   const char* name;
63.   const char* value;
64.} pubsub_opts_nameValue;
65.
66.//void usage(struct pubsub_opts* opts, const char* version, const char*
    program_name);
67.void usage(struct pubsub_opts* opts, pubsub_opts_nameValue* name_values,
    const char* program_name);
68.int getopts(int argc, char** argv, struct pubsub_opts* opts);
69.char* readfile(int* data_len, struct pubsub_opts* opts);
70.void logProperties(MQTTProperties *props);
71.
72.#endif
```

9. Описание результатов работы системы и анализ полученных данных: После запуска системы необходимо проверить, что данные из датчика DS18B20 успешно передаются в MQTT broker и сохраняются в .csv файл на сервере. Для этого можно использовать функции просмотра сообщений в MQTT broker и функции чтения файлов в Node-Red. Также можно использовать программы для анализа данных, такие как Microsoft Excel или LibreOffice Calc.

   Файлы, использованные в проекте, загружены в репозиторий https://github.com/ShurikSamarin/Coursework_15.07.24

Заключение.

1. Краткие теоретические и практические выводы, полученные во время разработки клиент-серверной платформы на базе Raspberry Pi для измерения температуры датчиком DS18B20, сохранением данных в .csv файл, передачей данных с помощью протокола MQTT на сервер в среду Node-Red и последующим сохранением данных на сервер в .csv файл:

В ходе разработки клиент-серверной платформы на базе Raspberry Pi для измерения температуры датчиком DS18B20 были получены следующие теоретические и практические выводы:

- Raspberry Pi является эффективным и доступным устройством для разработки клиент-серверных платформ для измерения температуры.
- Датчик DS18B20 поддерживает точные измерения температуры и имеет простую и доступную архитектуру подключения к GPIO-пинам Raspberry Pi.
- Протокол MQTT является эффективным инструментом для передачи данных между клиентом и сервером в реальном времени.
- Node-Red является удобным инструментом для обработки данных и создания визуальных интерфейсов для управления данными.
- Формат .csv является простым и доступным форматом для хранения данных.

2. Оценка проведённого исследования и достигнутых результатов:

Проведенное исследование позволило разработать клиент-серверную платформу для измерения температуры на базе Raspberry Pi с использованием датчика DS18B20, протокола MQTT, Node-Red и формата .csv. Были успешно решены задачи подключения датчика DS18B20 к Raspberry Pi, написания программы на Python для измерения температуры, настройки Raspberry Pi для передачи данных с использованием протокола MQTT, создания топика MQTT для передачи данных температуры, настройки Node-Red для приёма данных из топика MQTT и сохранения данных в .csv файл на сервере, тестирования и отладки системы, а также описания результатов работы системы и анализа полученных данных.

3. Практическая значимость работы и рекомендации по совершенствованию системы:

Разработанная клиент-серверная платформа для измерения температуры имеет практическую значимость для контроля и мониторинга температуры в различных областях, таких как промышленность, сельское хозяйство и медицина. Рекомендации по совершенствованию системы включают использование более точных датчиков температуры, увеличение частоты

измерений, добавление функции автоматического выключения датчика при достижении заданного предела температуры, создание веб-интерфейса для визуализации данных и управления системой, а также интеграцию системы с другими системами для более эффективного мониторинга и управления температурой.

4. Описание достижения цели проекта, выполнение задач и доказательство гипотезы:

Цель проекта заключалась в создании системы для измерения и мониторинга температуры в реальном времени на базе Raspberry Pi с использованием датчика DS18B20, протокола MQTT, Node-Red и формата .csv. Задачи проекта были успешно выполнены, что доказало гипотезу о возможности создания эффективной клиент-серверной платформы для измерения и мониторинга температуры на базе Raspberry Pi.

5. Предложения по совершенствованию системы для измерения температуры и обработки данных:

Для совершенствования системы для измерения температуры и обработки данных можно предложить следующие изменения:

- Использование более точных датчиков температуры для повышения точности измерений.
- Увеличение частоты измерений для более быстрого реагирования на изменения температуры.
- Добавление функции автоматического выключения датчика при достижении заданного предела температуры для предотвращения повреждения устройства.
- Создание веб-интерфейса для визуализации данных и управления системой.
- Интеграция системы с другими системами для более эффективного мониторинга и управления температурой.
- Разработка алгоритмов для анализа данных и выявления аномалий в температурных данных.
- Увеличение объема хранилища данных для сохранения большего объема данных в течение длительного периода времени.