

Знакомство с протоколом MQTT

Программирование на языке C
(IoT)



Оглавление

1. Инструментарий:	3
2. Цели семинара	3
План содержание:	4
Формат проведения семинара:	5
Разбор домашнего задания (при необходимости)	6
Вопросы по лекции	9
Задание 1	10
Клиент MQTT (MQTTClient)	10
Установка библиотек	10
Установка макросов программы	10
Инициализация структуры MQTTClient	11
Создание MQTT клиента	11
Основной цикл программы	12
Обработка ошибок подключения	12
Передача сообщения	12
Решение задания студентами	15
Задание 2	15
Подписчик MQTT ()	15
Иницизация структуры опций подключения	16
Функция int myconnect(MQTTClient client)	16
Основной цикл программы	17
Подписка к серверу	19
Решение задания студентами	21
Задание 3	21
Издатель	21
Подписчик	23
Решение задания студентами	25
Курсовая работа	26
Формат сдачи и критерии оценки курсовой работы	27
Ответы на вопросы студентов	27

1. Инструментарий:

- Методическое пособие и презентация по данной теме
- Демонстрация будет вестись на виртуальной машине ссылка на образ VM (Для VMware Workstation Player 17)

<https://files.iotserv.ru/downloadFile?id=MYN8OWxhwXcsqZ2>

Логин в систему (и доступ по ssh)

root/student1

Подключение к mosquitto брокеру порт 1883:

IoT/student1

Подключение к node-red порт 1880 через браузер:

admin/student1

Подключение к influxdb порт 8086 через браузер:

admin/student1

Подключение к grafana порт 3000 через браузер:

admin/student1

Тест брокера с VM

```
mosquitto_sub -h 127.0.0.1 -p 1883 -t GB -u "IoT" -P "student1"
```

```
mosquitto_pub -h 127.0.0.1 -p 1883 -t "GB" -m "Hello, GB!" -u "IoT" -P "student1"
```

(ну и с другой VM тоже можно через mosquitto clients и указание локального IP адреса VM)

https://github.com/Sudar1977/CforIOT_Lecture Демонстрации кода лекций
открытый (видно студентам)

https://github.com/Sudar1977/CforIOT_Demo Демонстрации кода семинаров
открытый (видно студентам)

https://github.com/Sudar1977/CforIOT_Practice Демонстрации кода практики
семинаров закрытый (не видно студентам)

https://github.com/Sudar1977/CforIOT_Homework Демонстрации кода домашних
заданий закрытый (не видно студентам)

2. Цели семинара

- Освоение применения протокола MQTT на практике через рассмотрение конкретных примеров его использования.
- Напишем собственное приложение, работающее по принципу «издатель - подписчик»
- Научимся рефакторить код приложения для повышения эффективности его работы и улучшения производительности.
- Получим задание на курсовую работу

План содержание:

Этап урока	Тайминг, минуты	Формат
Приветствие Разбор домашнего задания и ответы на вопросы студентов	10	Модерирует преподаватель
Викторина по пройденной лекции Опрос студентов по лекции, подготовка к семинару	10-15	Преподаватель зачитывает вопросы и разъясняет ответы. Выявить пробелы у студентов, спровоцировать у них вопросы
Демонстрация рабочей среды		Преподаватель демонстрирует рабочую среду, на примере которой будут показываться задания.
1-е Практическое задание для студентов	30	Модерирует преподаватель Преподаватель дает задание и объединяет студентов в команды
2-е Практическое задание для студентов	30	После завершения преподаватель показывает вариант решения, разбирает ошибки и отвечает на вопросы
3-е Практическое задание для студентов	30	
Подведение итогов	10-15	Преподаватель рассказывает, какие навыки были получены на семинаре. Что нужно знать и уметь по итогам семинара
Объяснение курсовой работы	5-10	Преподаватель дает пояснения к курсовой работе
Ответы на вопросы студентов	5	Преподаватель отвечает на

		вопросы студентов
Длительность:	130-145	

Формат проведения семинара:

1. Семинар проводится в формате митапа, когда руководитель показывает некоторые приемы программирования, раздает задания и дает обратную связь по готовым решениям.
2. Сначала преподаватель проводит небольшую викторину с теоретическими вопросами по пройденной лекции
3. Преподаватель расшаривает свой экран и проводит демонстрацию приемов программирования по заданной теме
4. Если студентов более четырех человек их нужно разделить на сессионные залы. Не более пяти залов. Для наилучшей организации групповой работы, желательно в каждый зал добавить студентов, которые хорошо разбираются в вопросе, и могут помочь сокурсникам.
5. Ученики получают задание, приступают к его выполнению
6. Студенты задают вопросы, если им что-то непонятно
7. Стоит обратить внимание на студентов, у которых не получается запустить программу, или есть технические неполадки или полное непонимание задания. По желанию студента можно попробовать решить проблему.
8. Дальше преподаватель в каждой группе проверяет выполнение задания. Представитель каждой группы расшаривает экран и показывает свой вариант решения. Если студентов мало, то можно посмотреть работы всех студентов.
9. Преподаватель собирает всех студентов в одну группу и показывает свой вариант решения и дает необходимые пояснения.
10. Переход к новому заданию

Разбор домашнего задания (при необходимости)

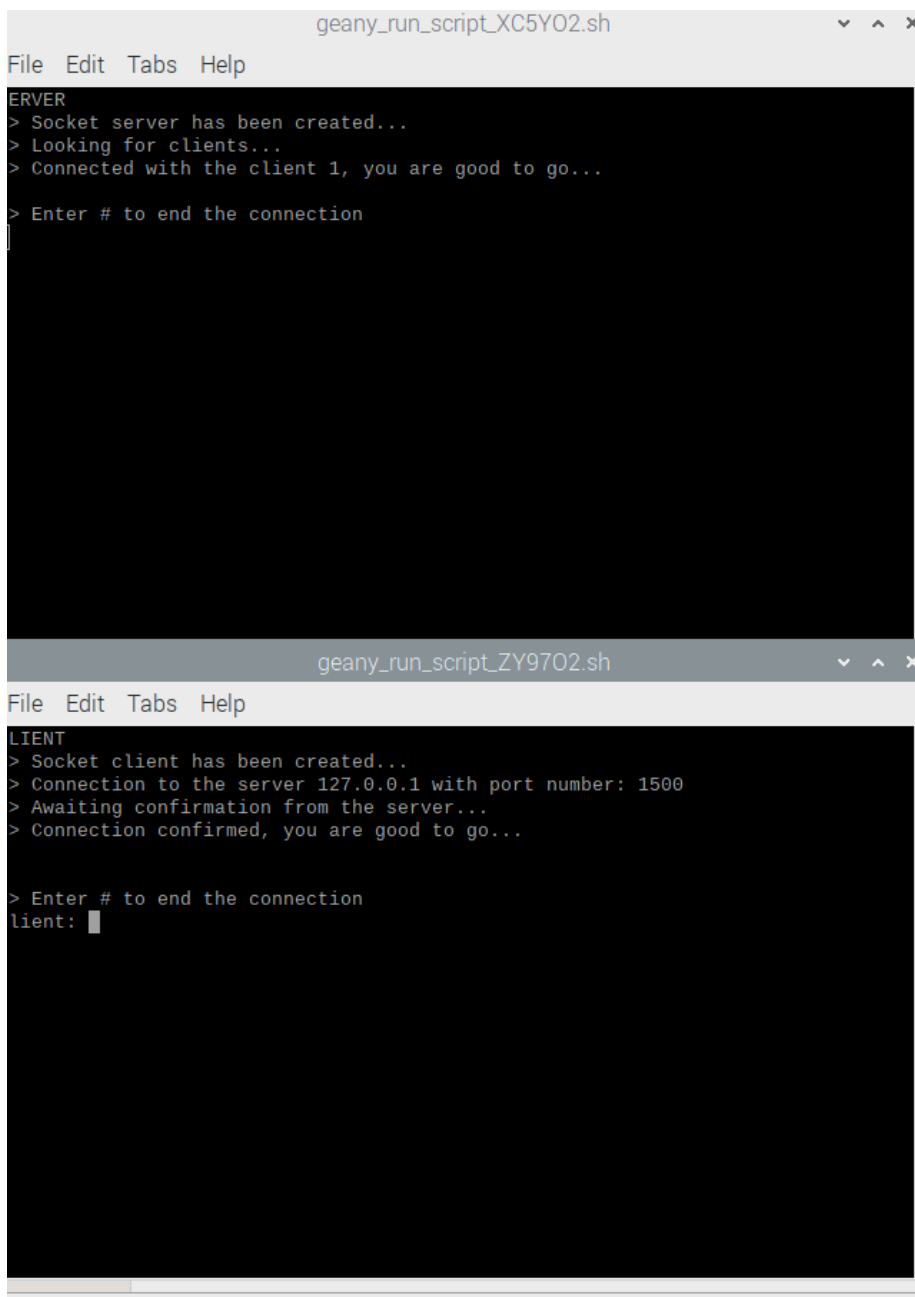
1. Написать приложение «Локальный чат» на архитектуре «клиент-сервер»,

Сообщение должно заканчиваться символом точка «.».

Вариант решения от преподавателя:

```
do {
    printf("Client: ");
    char c;
    int cntr = 0;
    while((c=getchar())!='\n')//till end of line
        buffer[cntr++]=c;
    buffer[cntr++] = 0;
    if (buffer[0] == '#')
        isExit = true;
    send(client, buffer, bufsize, 0);
    send(client, ".", 2, 0);

    printf("Server: ");
    do {
        recv(client, buffer, bufsize, 0);
        printf("%s ",buffer);
        if (buffer[0] == '#')
        {
            isExit = true;
        }
    } while (buffer[0] != '.' && buffer[0] != '#');
    printf("\n");
} while (!isExit);
```



The image shows two overlapping terminal windows. The top window, titled 'geany_run_script_XC5Y02.sh', displays the output of a server script. It shows the server being created, looking for clients, and successfully connecting to client 1. The bottom window, titled 'geany_run_script_ZY9702.sh', displays the output of a client script. It shows the client being created, connecting to the server at 127.0.0.1 on port 1500, and receiving confirmation from the server. Both windows have a menu bar with 'File', 'Edit', 'Tabs', and 'Help'.

```
geany_run_script_XC5Y02.sh
File Edit Tabs Help
SERVER
> Socket server has been created...
> Looking for clients...
> Connected with the client 1, you are good to go...
> Enter # to end the connection

geany_run_script_ZY9702.sh
File Edit Tabs Help
CLIENT
> Socket client has been created...
> Connection to the server 127.0.0.1 with port number: 1500
> Awaiting confirmation from the server...
> Connection confirmed, you are good to go...
> Enter # to end the connection
lient: #
```

2.* Написать приложение «Локальный чат» на архитектуре «клиент-сервер» с использованием байт-стаффинга (byte stuffing).

Escape (ESC) – конец подключения

End of message (EOM) – конец сообщения

Вариант решения от преподавателя:

```
//typedef enum {false, true} bool;

#define SOP 0x7E
```

```

#define EOP 0x7E
#define ESC 0x7D
#define XOR 0x20

//добавление байтстафинга 1 байта
int bytestuff(uint8_t *buf0,uint8_t byte,int len)
{
    switch(byte)
    {
        case SOP:
            buf0[len++] = ESC;
            buf0[len++] = SOP^XOR;
            break;
        case ESC:
            buf0[len++] = ESC;
            buf0[len++] = ESC^XOR;
            break;
        default:
            buf0[len++] = byte;
            break;
    }
    return len;
}

typedef enum {WAIT_SOF,ESC_SEQ,WAIT_EOF} STATE_RX;
//-----
//разборщик очищенного принятого пакета
void dispel_rx_buf0(uint8_t* buf0,int len)
{
    for(int i=0;i<len;i++)
        putc(buf0[i]);
}
//-----

//очитска принятого байта от байтстафинга
void dispel_rx_byte(uint8_t byte,uint8_t *buf)
{
    //Ответ на запрос статуса
    static STATE_RX state_rx = WAIT_SOF;
    int len = 0;
    switch(state_rx)
    {
        case WAIT_SOF:
            if(byte==SOP)
                state_rx=WAIT_EOF;
            len=0;
            break;
        case WAIT_EOF:
            switch(byte)
            {
                case ESC:
                    state_rx=ESC_SEQ;
                    break;
                case EOP:
                    dispel_rx_buf(buf0,len); //разборка пакета
                    state_rx=WAIT_SOF;
                    break;
                default:
                    buf0[len++] = byte;
                    break;
            }
            break;
        case ESC_SEQ:

```



```

        switch(byte)
        {
            case SOP^XOR:
                buf0[len++] = SOP;
                state_rx=WAIT_EOF;
                break;
            case ESC^XOR:
                len++;
                buf0[len] = ESC;
                state_rx=WAIT_EOF;
                break;
        }
        break;
    }
}

```

3. Используя практическое задание №2 необходимо «разобрать» кнопку, т.е. отправлять в консоль данные ее нажатия.

Вариант решения от преподавателя:

```

#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    /* ----- ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕННЫХ СЕРВЕРА----- */

    /*
    1. client/server - два файл-дескриптора.
    Эти две переменные хранят значение сокетов,
    которые вернула система при подключении.
    2. portNum нужен для хранения номера порта, на котором происходит
    соединение.
    3. isExit - булевая переменная, признак завершения программы.
    4. Сервер считывает сообщение из сокет соединения в буффер обмена для
    приема/отправки данных к/от сервера.

    */
    int server;//файл-дескриптор сервера
    int client;//файл-дескриптор клиента
    int portNum = 8000;//номера порта, на котором происходит соединение (0 до
65535)
    bool isExit = false;//булевая переменная, признак завершения программы.
    int bufsize = 1024;//размер буффера
    char buffer[bufsize]; //буффер обмена для приема/отправки данных к/от
сервера
    /*

```

```

5. А sockaddr_in - структура, содержащая интернет адрес, с которым будет
установлено соединение.
Эта структура уже определена в netinet/in.h, поэтому нет необходимости
заново ее задавать.
DEFINITION:
struct sockaddr_in
{
    short    sin_family;
    u_short  sin_port;
    struct    in_addr sin_addr;
    char     sin_zero[8];
};
6. serv_addr будет содержать адрес сервера
7. socklen_t - длиной по крайней мере 32 бита
*/

struct sockaddr_in server_addr;
socklen_t size;

/* ----- УСТАНОВКА СОКЕТ СОЕДИНЕНИЯ -----*/
/* ----- socket() функция -----*/

/*
Socket() функция создает новый сокет.
На вход получает 3 аргумента,
    а. AF_INET: доменный адрес сокета.
    б. SOCK_STREAM: тип сокета. Поточный сокет, в котором сообщение
читается последовательным потоком (TCP).
    в. Третий это аргумент протокола: всегда должен быть 0.
Операционная система выберет самый подходящий протокол.
Функция возвращает файл-дескриптор целое число - соединение,
которое используется для всех ссылок
на этот сокет. Если произошла ошибка соединения, то возвращается
-1.
*/

server = socket(AF_INET, SOCK_STREAM, 0);
printf("SERVER\n");

if (server < 0)
{
    printf("Error establishing socket...\n");
    exit(1);
}

printf("=> Socket server has been created...\n");

/*
Переменная serv_addr - структура sockaddr_in.
sin_family содержит код для адресной семьи.
Всегда должна быть установлена AF_INET.
INADDR_ANY содержит IP-адрес хоста. Для сервера - это
IP-адрес компьютера, на котором работает сервер.
htons() функция переводит номер порта из порядка байтов хоста
в номер порта в порядке байтов сети.
*/

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(portNum);

/* ----- ПРИВЯЗКА СОКЕТА ----- */
/* ----- bind() ----- */

```

```

/*
    bind() функция привязывает сокет к адресу, то есть в нашем случае
    к адресу сервера и номеру порта, на котором сервер будет работать.
    Для этого нужны три аргумента: файл дескриптор сокета, указатель на
структуру
    типа sockaddr (должен указывать на правильный тип)
    */
*/

if ((bind(server, (struct sockaddr*)&server_addr, sizeof(server_addr))) <
0)
{
    printf("=> Error binding connection, the socket has already been
established...\n");
    return -1;
}

size = sizeof(server_addr);
printf("=> Looking for clients...\n");

/* ----- ПРОСЛУШИВАНИЕ СОКЕТА ----- */
/* ----- listen() ----- */

/*
    Функция listen позволяет прослушивать сокета для подключения.
    Программа будет в состоянии простоя, если подключений не обнаружится.
    Первый аргумент - это файл дескриптор сокета, второй - количество
клиентов,
    то есть количество подключений, которое сервер может обработать, пока
процесс
    обрабатывает определенное подключение. Максимальное число клиентов во
многих системах равняется 5.
    */

listen(server, 1);

/* ----- ПОДКЛЮЧЕНИЕ КЛИЕНТОВ ----- */
/* ----- accept() ----- */

/*
    Система accept() вызывает процесс блокировки пока клиент подключается к
серверу.
    К тому же, она пробуждает процесс, когда подключение с клиентом было
успешно установлено.
    Она возвращает новый файл дескриптор, и вся связь по этому подключению
должна
    осуществляться, используя новый файл дескриптор. Второй аргумент -
указатель на
    адрес клиента, третий - размер структуры.
    */

int clientCount = 1;

while(!isExit)
{
    client = accept(server, (struct sockaddr *)&server_addr, &size);

    // первая проверка действительности подключения
    if (client < 0)

```

```

        printf("=> Error on accepting...");

//Основной цикл
    if(client > 0)
    {
        printf("=> Connected with the client %d, you are good to
go...\n",clientCount);
        //.....

        /*
        Примечание: мы перейдем в этот момент только после того, как
клиент успешно подключится к нашему серверу.
Произойдет чтение сокета. Заметим, что read() будет
блокировать до момента наличия чего-то для чтения
Чтение будет происходить maximum 1024 символов в пакете
*/

        int result = recv(client, buffer, bufsize, 0);
        if (result < 0)
        {
            // ошибка получения данных
            printf("\n\n=> Connection terminated error %d with IP
%s\n",result,inet_ntoa(server_addr.sin_addr));
            close(client);
            exit(1);
        }
        // Мы знаем фактический размер полученных данных, поэтому ставим
метку конца строки
        // В буфере запроса.
        buffer[result] = '\0';
        char response[1024] = "HTTP/1.1 200 OK\r\n"
"Version: HTTP/1.1\r\n"
"Content-Type: text/html; charset=utf-8\r\n"
"\r\n\r\n"
"<!DOCTYPE HTML>"
"<html>"
"  <head>"
"    <meta name=\"viewport\" content=\"width=device-width,\"
"      initial-scale=1\">"
"  </head>"
"  <h1>ESP32 - Web Server</h1>"
"  <p>LED #1"
"    <a href=\"/on1\">"
"      <button>ON</button>"
"    </a>&nbsp;"
"    <a href=\"/off1\">"
"      <button>OFF</button>"
"    </a>"
"  </p>"
"  <p>LED #2"
"    <a href=\"/on2\">"
"      <button>ON</button>"
"    </a>&nbsp;"
"    <a href=\"/off2\">"
"      <button>OFF</button>"
"    </a>"
"  </p>";
        strcat(response,"</html>");
        printf("%s\n",buffer);
        char* get_str_on1 = strstr(buffer,"/on1");
        char* get_str_on2 = strstr(buffer,"/on2");
        char* get_str_off1 = strstr(buffer,"/off1");
        char* get_str_off2 = strstr(buffer,"/off2");
        if(get_str_on1)
            printf("%s\n",get_str_on1);

```

```

        if (get_str_on2)
            printf("%s\n",get_str_on2);
        if (get_str_off1)
            printf("%s\n",get_str_off1);
        if (get_str_off2)
            printf("%s\n",get_str_off2);

        send(client, response,strlen(response), 0);
        printf("\n\n=> Connection terminated with IP
%s\n",inet_ntoa(server_addr.sin_addr));
        close(client);
        printf("\n=> Press any key and <Enter>, # to end the
connection\n");
        char c;
        while((c=getchar())!='\n')
            if(c=='#')
            {
                isExit=true;
            }
    }
    close(server);
    printf("\nGoodbye...");
    isExit = false;
    return 0;
}

```

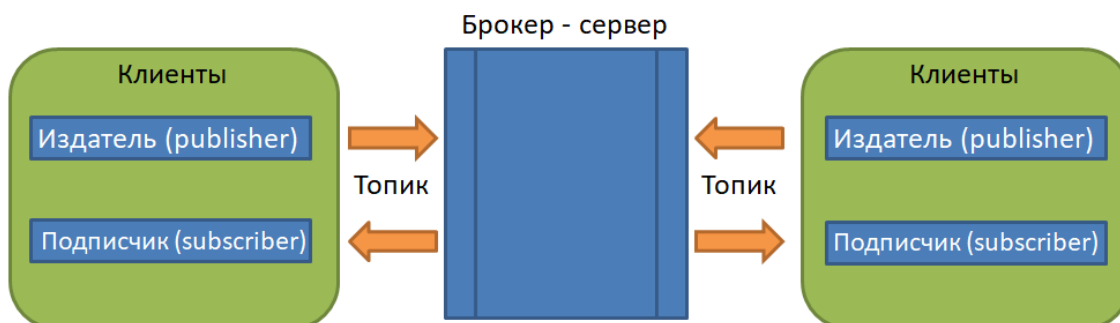
Вопросы по лекции

1. Что представляет из себя протокол обмена MQTT?

MQTT (Message Queue Telemetry Transport) – протокол обмена сообщениями, реализуемый по принципу «издатель-подписчик» («отправитель-получатель»).

Взаимодействие между издателями и подписчиками обеспечивает третий компонент – брокер сообщений. Задача брокера – отфильтровать сообщения от издателей и отправить их соответствующим подписчикам.

Брокер действует как почтовое отделение. Клиенты MQTT не используют адрес прямого подключения предполагаемого получателя, но используют строку темы, называемую **тема** или **topic**.



2. Какие данные можно передавать по протоколу MQTT? Где используется протокол MQTT?

Формат полезных данных, передаваемых по протоколу MQTT, не стандартизован, поэтому может быть закодирован в любом формате: XML, JSON и т.д.

В наше время данный протокол активно используется в системах умного дома (IoT) для передачи данных от различных датчиков, для управления лампами, розетками и т.д.

3. Какая структура у управляющих пакетов? Какие бывают заголовки?

Управляющие пакеты имеют заголовок и полезную нагрузку. Причём, в некоторых пакетах может быть два заголовка – фиксированный и переменный.

То есть, во всех пакетах MQTT присутствует фиксированный заголовок, а переменный заголовок и полезная нагрузка могут быть не во всех.

4. Какие бывают библиотеки?

В языке Си библиотеки могут существовать в двух вариациях:

- **статические**: код из библиотеки добавляется в исполняемый файл на стадии

линковки, и после её окончания файл библиотеки больше не нужен полученной программе;

• **динамические:** код из библиотеки не добавляется в исполняемый файл, а загружается в память во время запуска программы. Таким образом, он должен быть доступен при каждом запуске.

Задание 1

Тайминг – 20 мин

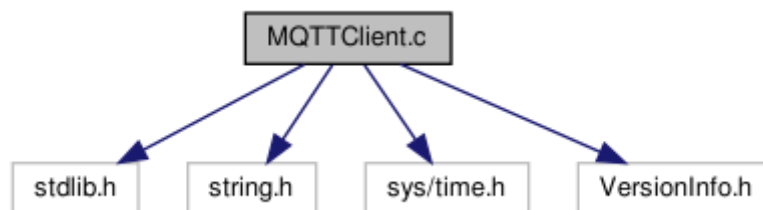
Клиент MQTT (MQTTClient)

Напишем своего клиента сервера MQTT, используя сетевой IP-адрес.

Для написания своего клиента MQTT нам нужно выполнить шаги:

- [Установка библиотек](#)
- [Установка макросов программы](#)
- [Инициализация структуры MQTTClient](#)
- [Создание MQTT клиента](#)
- [Основной цикл программы](#)
- [Обработка ошибок подключения](#)
- [Передача сообщения](#)

Установка библиотек



```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "MQTTClient.h"
```

Установка макросов программы

- ADDRESS – адрес сервера MQTT
- CLIENTID – идентификатор клиента
- TOPIC – топик MQTT – символы с кодировкой UTF-8
- PAYLOAD – передаваемое сообщение
- QOS (quality of service) – уровень качества обслуживания (0,1,2)
- TIMEOUT – задержка отправки сообщения

QOS показывает вероятность прохождения пакета между двумя точками сети. Оно бывает:

- QOS 0 – не более одного раза, сервер отправляет и забывает. Сообщения могут быть потеряны или продублированы. Самый быстродействующий уровень, но ненадежный
- QOS 1 – по крайней мере один раз, получатель подтверждает доставку. Сообщения могут дублироваться, но доставка гарантирована. Уровень по умолчанию.
- QOS 2 – ровно один раз, сервер обеспечивает доставку. Сообщения поступают точно один раз без потери или дублирования. Самый надежный уровень, но наиболее медленный.

```
#define ADDRESS      "mqtt.eclipseprojects.io:1883"
#define CLIENTID     "ExampleClientPub"
#define TOPIC        "MQTT Examples"
#define PAYLOAD      "Hello World!"
#define QOS          1
#define TIMEOUT      10000L
```

Инициализация структуры MQTTClient

- MQTTClient устанавливает клиента соединения
- MQTTClient_connectOptions определяет параметры, которые управляют способом подключения клиента к MQTT серверу.
- MQTTClient_message – структура, представляющая сообщение и его атрибуты
- MQTTClient_deliveryToken задает определенный токен сообщению

```
MQTTClient client;
MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
MQTTClient_message pubmsg = MQTTClient_message_initializer;
MQTTClient_deliveryToken token;

int rc;
```

Создание MQTT клиента

Функция MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL) создает MQTT клиента, готового к соединению к определенному серверу.

Требуется 5 аргументов:

- Указатель на клиента соединения
- Адрес сервера MQTT
- Идентификатор клиента
- Тип сохранения, который будет использовать клиент: в памяти (MQTTCLIENT_PERSISTENCE_NONE), по умолчанию (MQTTCLIENT_PERSISTENCE_DEFAULT), конкретного приложения (MQTTCLIENT_PERSISTENCE_USER)
- Путь сохранения. При использовании MQTTCLIENT_PERSISTENCE_NONE этот аргумент не используется и должен быть присвоен NULL.

`conn_opts.keepAliveInterval` – определяет максимальное время, которое может пройти без «общения» клиента и сервера. Измеряется в секундах.

`conn_opts.cleansession` контролирует поведение клиента и сервера в момент подключения и отключения. Обеспечивает «по крайней мере одну» и «только одну» отправку сообщения, а также «только один» прием.

```
MQTTClient_create(&client, ADDRESS, CLIENTID,  
    MQTTCLIENT_PERSISTENCE_NONE, NULL);  
conn_opts.keepAliveInterval = 20;  
conn_opts.cleansession = 1;
```

Основной цикл программы

Обработка ошибок подключения

Если произошла ошибка соединения, то возвращается -1 и на консоль выводится соответствующая ошибка.

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS)  
{  
    printf("Failed to connect, return code %d\n", rc);  
    exit(-1);  
}
```

```
pubmsg.payload = PAYLOAD;  
pubmsg.payloadlen = strlen(PAYLOAD);  
pubmsg.qos = QOS;  
pubmsg.retained = 0;
```

Передача сообщения

`MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token)` пробует опубликовать сообщение с присвоенным топиком. Использует 4 переменные:

- Действительный клиент соединения
- Присвоенный топик
- Указатель на действительную структуру `MQTTClient_Message`, содержащую сообщение и его атрибуты
- Указатель на токен сообщения

`MQTTClient_waitForCompletion(client, token, TIMEOUT)` блокирует выполнение программы пока сообщение не будет успешно доставлено или не пройдет таймаут системы. Требуется 3 переменные:

- Действительный клиент соединения
- Токен сообщения

- Таймаут

`MQTTClient_disconnect(client, 10000)` пробует отключить клиента от MQTT сервера. Использует действительный клиент соединения и значение таймаута системы. По истечению таймаута клиент отключится, даже если еще будут существовать данные для передачи.

`MQTTClient_destroy(&client)` освобождает память, выделенную MQTT серверу. Должна вызываться в момент, когда клиент больше не используется.

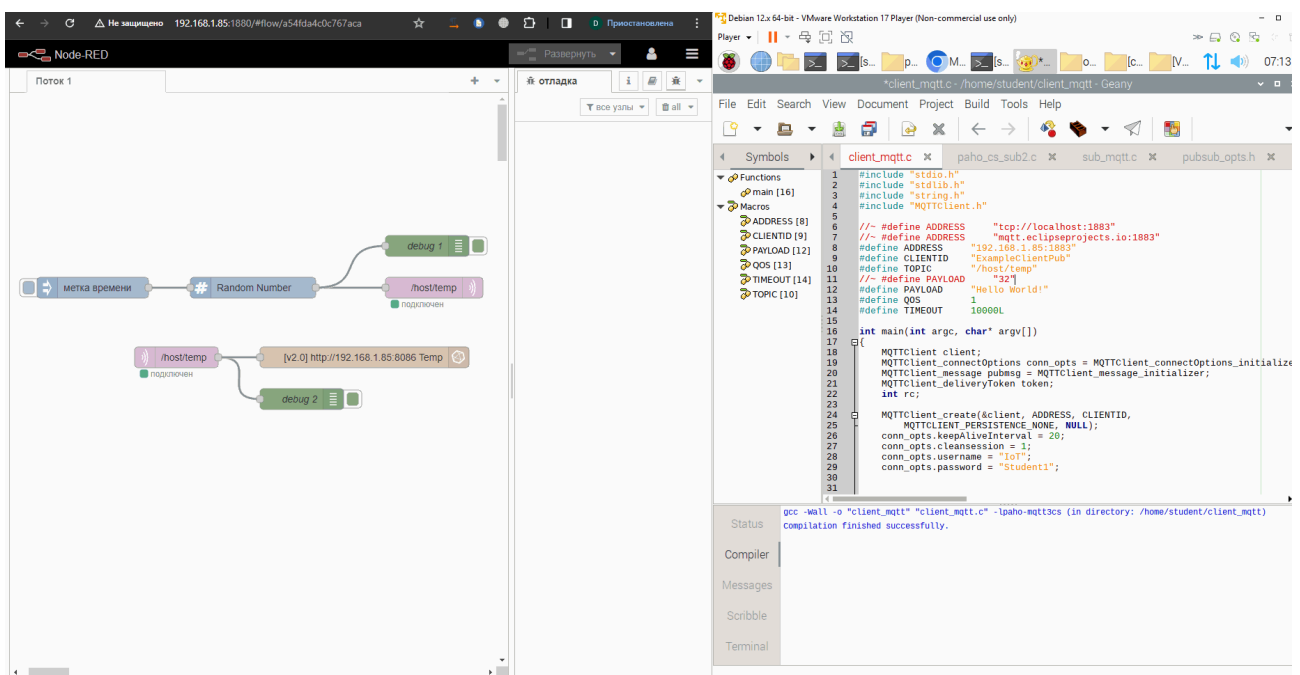
```
MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
printf("Waiting for up to %d seconds for publication of %s\n"
      "on topic %s for client with ClientID: %s\n",
      (int) (TIMEOUT/1000), PAYLOAD, TOPIC, CLIENTID);
rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
printf("Message with delivery token %d delivered\n", token);
MQTTClient_disconnect(client, 10000);
MQTTClient_destroy(&client);
return rc;
```

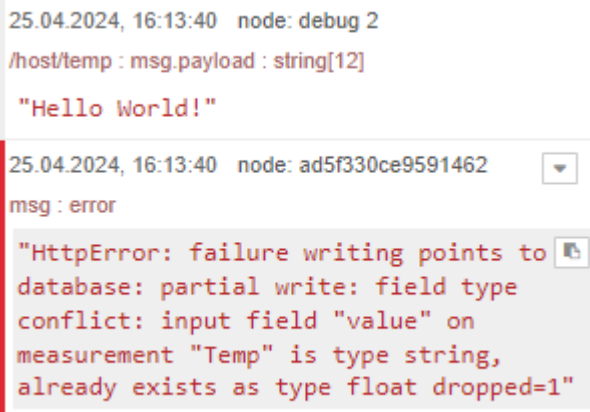
Результат работы программы:

Отправка сообщения прошла успешно.

В графической среде NodeRed мы видим полученное сообщение.

Однако, возникает ошибка содержания строки.



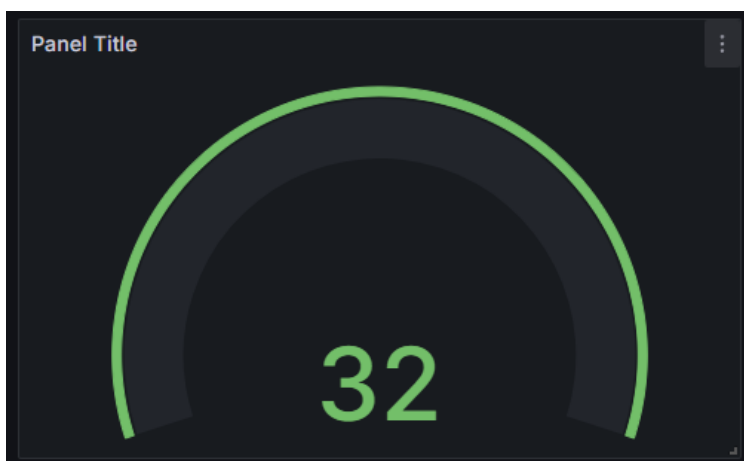
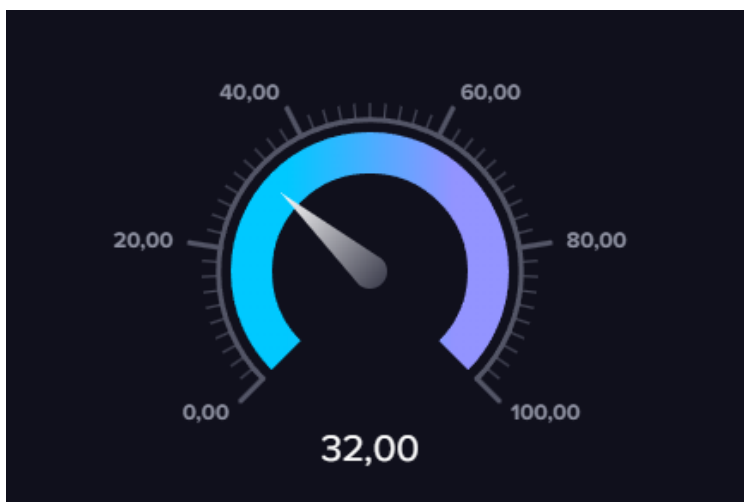


```
#define PAYLOAD      "32"
```

The screenshot displays a development environment with three main components:

- Node-RED (Left):** A visual programming interface showing a flow with nodes for "метка времени" (timestamp), "Random Number", "/host:temp", and "debug 1". The flow is connected to a "v2.0] http://192.168.1.85:8086 Temp" node, which is then connected to "debug 2".
- Terminal (Bottom Left):** A terminal window showing the command `gcc -Wall -o "client_mqtt" "client_mqtt.c" -lpaho-mqtt3cs` and the output `compilation finished successfully.`
- Code Editor (Right):** A code editor showing the C source file `client_mqtt.c`. The code includes headers for `stdio.h`, `stdlib.h`, `string.h`, and `MQTTClient.h`. It defines constants for `ADDRESS`, `CLIENTID`, `PAYLOAD`, `QOS`, `TIMEOUT`, and `TOPIC`. The `main` function initializes an `MQTTClient` and connects to a broker at `tcp://localhost:1883`.

```
25.04.2024, 16:14:14 node: debug 2
/host/temp : msg.payload : number
```



Решение задания студентами

Тайминг – 10 минут

Студентам предлагается проделать практическое задание №1 самостоятельно.

Задание 2

Тайминг – 20 мин

Подписчик MQTT ()

Для написания своего подписчика MQTT нам нужно выполнить шаги:

- [Инициализировать структура опций подключения](#)
- [Функция int myconnect\(MQTTClient client\)](#)
- [Основной цикл программы](#)
- [Подписка к серверу](#)

Инициализация структуры опций подключения

Устанавливаем в графы topic, host, port, user, password необходимые параметры подключения.

```
struct pubsub_opts opts =
{
    0, 0, 0, 0, "\n", 100, /* debug/app options */
    NULL, NULL, 1, 0, 0, /* message options */
    //~ MQTTVERSION_DEFAULT, NULL, "paho-cs-sub", 0, 0, NULL, NULL, "localhost",
    "1883", NULL, 10, /* MQTT options */
    MQTTVERSION_DEFAULT, "/host/temp", "paho-cs-sub", 0, 0, "IoT", "Student1",
    "192.168.1.85", "1883", NULL, 10, /* MQTT options */
    NULL, NULL, 0, 0, /* will options */
    0, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, /* TLS options */
    0, {NULL, NULL}, /* MQTT V5 options */
    NULL, NULL, /* HTTP and HTTPS proxies */
};
```

Функция `int myconnect(MQTTClient client)`

Установка параметров

`MQTTClient_connectOptions` определяет параметры, которые управляют способом подключения клиента к MQTT серверу.

`MQTTClient_SSLOptions` определяет параметры установки SSL/TLS подключения, используя библиотеку OpenSSL.

`MQTTClient_willOptions` определяет настройки MQTT “Last Will and Testament(LWT)” (Завещание) клиента.

```
MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
MQTTClient_SSLOptions ssl_opts = MQTTClient_SSLOptions_initializer;
MQTTClient_willOptions will_opts = MQTTClient_willOptions_initializer;

int rc = 0;
```

Проверка параметров подключения

```
if (opts.verbose)
    printf("Connecting\n");

if (opts.MQTTVersion == MQTTVERSION_5)
{
    MQTTClient_connectOptions conn_opts5 =
MQTTClient_connectOptions_initializer5;
    conn_opts = conn_opts5;
}

conn_opts.keepAliveInterval = opts.keepalive;
conn_opts.username = opts.username;
conn_opts.password = opts.password;
conn_opts.MQTTVersion = opts.MQTTVersion;
conn_opts.httpProxy = opts.http_proxy;
conn_opts.httpsProxy = opts.https_proxy;

if (opts.will_topic) /* will options */
{
    will_opts.message = opts.will_payload;
    will_opts.topicName = opts.will_topic;
    will_opts.qos = opts.will_qos;
}
```

```

will_opts.retained = opts.will_retain;
conn_opts.will = &will_opts;
}

if (opts.connection && (strncmp(opts.connection, "ssl://", 6) == 0 ||
    strncmp(opts.connection, "wss://", 6) == 0))
{
    if (opts.insecure)
        ssl_opts.verify = 0;
    else
        ssl_opts.verify = 1;
    ssl_opts.CApath = opts.capath;
    ssl_opts.keyStore = opts.cert;
    ssl_opts.trustStore = opts.cafile;
    ssl_opts.privateKey = opts.key;
    ssl_opts.privateKeyPassword = opts.keypass;
    ssl_opts.enabledCipherSuites = opts.ciphers;
    conn_opts.ssl = &ssl_opts;
}

if (opts.MQTTVersion == MQTTVERSION_5)
{
    MQTTProperties props = MQTTProperties_initializer;
    MQTTProperties willProps = MQTTProperties_initializer;
    MQTTResponse response = MQTTResponse_initializer;

    conn_opts.cleanstart = 1;
    response = MQTTClient_connect5(client, &conn_opts, &props, &willProps);
    rc = response.reasonCode;
    MQTTResponse_free(response);
}

```

Подключение к серверу

```

else
{
    conn_opts.cleansession = 1;
    rc = MQTTClient_connect(client, &conn_opts);
}

```

Проверка соединения

Если произошла ошибка соединения, то на консоль выводится соответствующая ошибка.

```

if (opts.verbose && rc == MQTTCLIENT_SUCCESS)
    fprintf(stderr, "Connected\n");
else if (rc != MQTTCLIENT_SUCCESS && !opts.quiet)
    fprintf(stderr, "Connect failed return code: %s\n",
MQTTClient_strerror(rc));

return rc;
}

```

Основной цикл программы

Инициализация структуры MQTTClient

- MQTTClient устанавливает клиента соединения

- MQTTClient_connectOptions определяет параметры, которые управляют способом подключения клиента к MQTT серверу.
- MQTTClient_createOptions создает клиента MQTT, готового к подключению на определенный сервер

```
int main(int argc, char** argv)
{
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_createOptions createOpts = MQTTClient_createOptions_initializer;
    int rc = 0;
    char* url;
    const char* version = NULL;
#ifdef _WIN32
    struct sigaction sa;
#endif
    const char* program_name = "paho_cs_sub";
    MQTTClient_nameValue* infos = MQTTClient_getVersionInfo();
```

Установка параметров

```
/*
    if (argc < 2)
        usage(&opts, (pubsub_opts_nameValue*)infos, program_name);

    if (getopts(argc, argv, &opts) != 0)
        usage(&opts, (pubsub_opts_nameValue*)infos, program_name);
*/
if (strchr(opts.topic, '#') || strchr(opts.topic, '+'))
    opts.verbose = 1;

if (opts.connection)
    url = opts.connection;
else
{
    url = malloc(100);
    sprintf(url, "%s:%s", opts.host, opts.port);
}
if (opts.verbose)
    printf("URL is %s\n", url);

if (opts.tracelevel > 0)
{
    MQTTClient_setTraceCallback(trace_callback);
    MQTTClient_setTraceLevel(opts.tracelevel);
}

if (opts.MQTTVersion >= MQTTVERSION_5)
    createOpts.MQTTVersion = MQTTVERSION_5;
rc = MQTTClient_createWithOptions(&client, url, opts.clientid,
MQTTCLIENT_PERSISTENCE_NONE,
    NULL, &createOpts);
if (rc != MQTTCLIENT_SUCCESS)
{
    if (!opts.quiet)
        fprintf(stderr, "Failed to create client, return code: %s\n",
MQTTClient_strerror(rc));
    exit(EXIT_FAILURE);
}
```

Проверка подключения

Если произошла ошибка соединения, то процесс выполнения программы завершается.

```
if (myconnect(client) != MQTTCLIENT_SUCCESS)
    goto exit;
```

Подписка к серверу

MQTTClient_subscribe(client, opts.topic, opts.qos) пробует подписать клиента к определенному топику. Требуется 3 аргумента:

- Действительный клиент соединения
- Топик
- QOS

Если произошла ошибка соединения, то выполнение программы завершается.

```
if (opts.MQTTVersion >= MQTTVERSION_5)
{
    MQTTResponse response = MQTTClient_subscribe5(client, opts.topic,
opts.qos, NULL, NULL);
    rc = response.reasonCode;
    MQTTResponse_free(response);
}
else
    rc = MQTTClient_subscribe(client, opts.topic, opts.qos);
if (rc != MQTTCLIENT_SUCCESS && rc != opts.qos)
{
    if (!opts.quiet)
        fprintf(stderr, "Error %d subscribing to topic %s\n", rc, opts.topic);
    goto exit;
}
```

Цикл работы программы

MQTTClient_receive(client, &topicName, &topicLen, &message, 1000) производит синхронное получение входящих сообщений. Функции необходимо 5 параметров:

- Действительный клиент соединения
- Указатель на топик
- Указатель на длину топика
- Указатель на полученное сообщение
- Таймаут

```
while (!toStop)
{
    char* topicName = NULL;
    int topicLen;
    MQTTClient_message* message = NULL;

    rc = MQTTClient_receive(client, &topicName, &topicLen, &message, 1000);
    if (rc == MQTTCLIENT_DISCONNECTED)
        myconnect(client);
    else if (message)
    {
        size_t delimlen = 0;
```



```

        if (opts.verbose)
            printf("%s\t", topicName);
        if (opts.delimiter)
            delimlen = strlen(opts.delimiter);
        if (opts.delimiter == NULL || (message->payloadlen > delimlen &&
            strncmp(opts.delimiter,
&((char*)message->payload)[message->payloadlen - delimlen], delimlen) == 0))
            printf("%.s", message->payloadlen, (char*)message->payload);
        else
            printf("%.s%s", message->payloadlen, (char*)message->payload,
opts.delimiter);
//        if (message->struct_version == 1 && opts.verbose)
//            logProperties(&message->properties);
fflush(stdout);
MQTTClient_freeMessage(&message);
MQTTClient_free(topicName);
    }
}

```

Завершение соединения

`MQTTClient_disconnect(client, 0)` пробует отключить клиента от MQTT сервера. Использует действительный клиент соединения и значение таймаута системы. По истечению таймаута клиент отключится, даже если еще будут существовать данные для передачи.

`MQTTClient_destroy(&client)` освобождает память, выделенную MQTT серверу. Должна вызываться в момент, когда клиент больше не используется.

```

exit:
    MQTTClient_disconnect(client, 0);

    MQTTClient_destroy(&client);

    return EXIT_SUCCESS;

```

Результат работы программы:

The image displays the development environment for the MQTT client. On the left, the Node-RED web interface shows a flow with a 'метка времени' (time marker) node, a 'Random Number' node, and an 'HTTP GET' node connected to 'http://192.168.1.85:8086/Temp'. On the right, a C code editor shows the source code for 'paho_cs_sub2.c'. The code includes MQTT client initialization, connection, and subscription functions. The status bar at the bottom indicates 'compilation finished successfully'.

Решение задания студентами

Тайминг – 10 минут

Студентам предлагается проделать практическое задание №2 самостоятельно.

Задание 3

Тайминг – 20 мин

Разработаем визуальную составляющую датчика температуры в системе «умный дом».

Датчик записывает в отдельный файл массив данных, состоящий из даты измерения и величину температуры.

Издатель считывает текстовый файл и отправляет его в среду разработки Node-Red по протоколу обмена MQTT через фиксированный временной интервал.

Подписчик подписан на среду Node-Red и считывает значения датчика. Подписчик записывает полученные данные в файл с указанием времени и даты записи.

Издатель

Создаем структуру датчика температуры

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

Для записи показаний термометра, а также даты записи необходимо создать функцию записи данных

```
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;  
}
```

Необходимо не забыть выделить память под структуру

```
struct sensor*info = malloc (365*24*60*sizeof(struct sensor));
```

В основном теле программы нужно открыть файл с показаниями датчика. Для этого воспользуемся функцией `fopen()`. Формат `"r"` – открыть файл только для чтения (файл должен существовать).

```
FILE *file;  
file = fopen("temperature_small1.csv","r");
```

Создадим цикл для считывания и записи информации из текстового файла, в котором будет реализована проверка на корректность введенных данных

```
for (;(r=fscanf(file,"%d;%d;%d;%d;%d;%d",&Y,&M,&D,&H,&Min,&T))>0;count++)  
{  
    if (r<6)  
    {  
        char s[20], c;  
        r = fscanf (file, "%[^\n]%c",s,&c);  
        printf("Wrong format in line %s\n",s);  
    }  
    else  
    {  
        printf("%d %d %d %d %d %d\n",Y,M,D,H,Min,T);  
        AddRecord(info,count,Y,M,D,H,Min,T);  
    }  
}  
fclose(file);
```

По завершении работы с файлом его необходимо закрыть функцией `fclose()`.

Для отправки данных температуры в графическую среду Node-Red нужно перевести формат данных из `%d` (целое со знаком и в десятичном виде) в `%s` (строка – массив букв). Для этой цели используем функцию `sprintf()`

```
char str[255];  
sprintf(str,"%d",info[i++].t);  
printf("%s,%d\n",str,i);  
  
if(i>=count)  
    i=0;
```

Теперь вместо `payload` в сообщении нужно подсунуть наше считанное значение температуры и отправить его по протоколу обмена MQTT.

```
pubmsg.payload = str;  
pubmsg.payloadlen = strlen(str);  
pubmsg.qos = QOS;  
pubmsg.retained = 0;  
MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);  
rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
```

Для обеспечения отправки через фиксированный промежуток времени в тело цикла необходимо добавить 2 функции времени – в начало и в конец. Скорость отправки определяется значением параметра `double DELAY = 5.`

```
clock_t begin = clock();
```

```
while ((double)(clock() - begin)/CLOCKS_PER_SEC<DELAY)
{
}
```

Результат работы издателя:

Данные приходят в Node-Red.

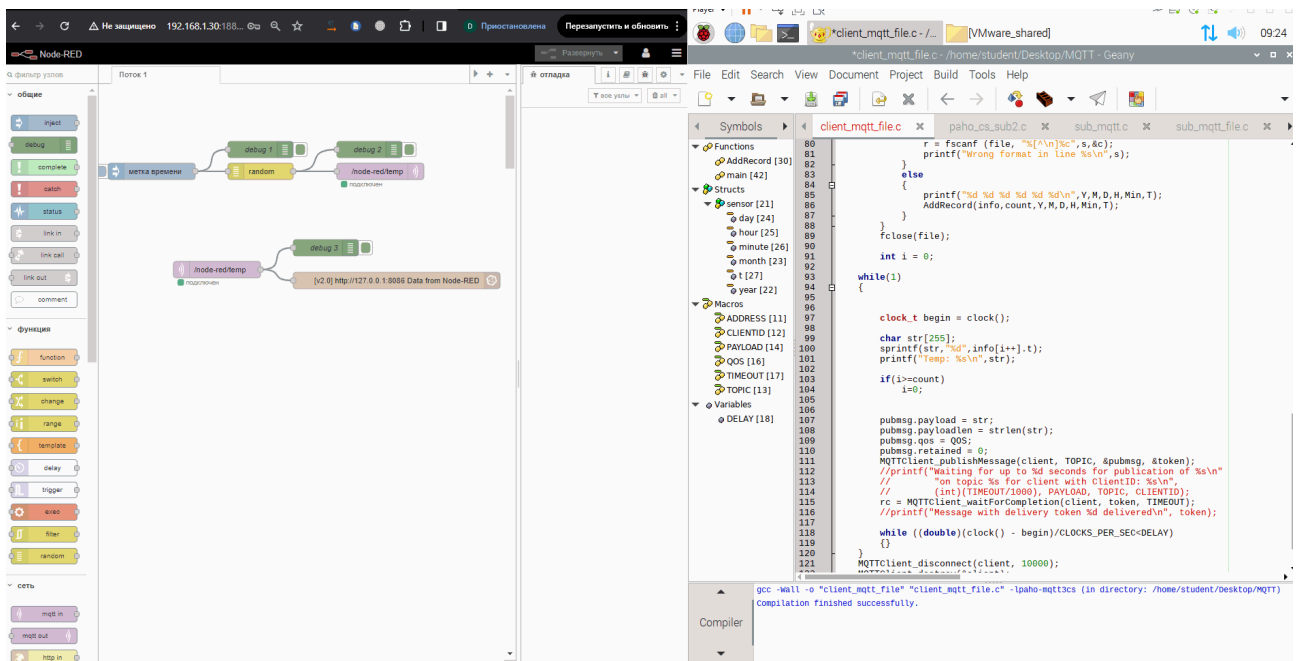


График InfluxDB



Подписчик

В основном теле программы нужно открыть файл, в который будут записаны показания датчика. Для этого воспользуемся функцией `fopen()`. Формат `"w"` – открыть пустой файл для записи; если файл существует, то его содержимое теряется.

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

Для получения реальной даты и времени необходимо подключить библиотеку time.h.

```
#include "time.h"
```

Заголовочный файл определяет функции для работы с датой и временем. В нашем случае будет использоваться функция time, которая возвращает текущие дату и время в виде объекта типа time_t.

Чтобы получить дату/время и ее компоненты (часы, минуты и т.д.), нам надо получить из объекта time_t структуру tm с помощью функции localtime()

```
time_t mytime = time (NULL);  
struct tm *now = localtime(&mytime);
```

Структура tm хранит данные в ряде своих элементов, каждый из которых представляет тип int:

- tm_sec – секунды от 0 до 60
- tm_min – минуты от 0 до 59
- tm_hour – часы от 0 до 23
- tm_mday – день месяца от 1 до 31
- tm_mon – месяц от 0 до 11
- tm_year – год, начиная с 1900
- tm_wday – день недели от 0 до 6 (воскресение имеет номер 0)
- tm_yday – количество дней года, прошедших с 1 января, от 0 до 365
- tm_isdst – если больше 0, то установлен переход на летнее время. Если равно 0, то переход на летнее время не действует.

Функция strftime() преобразует дату и время в строку в определенном формате

```
char time[20];  
char date[20];  
  
strftime(date, sizeof(str), "%D", now);  
strftime(time, sizeof(str), "%T", now);
```

Она принимает четыре значения:

- Строка, в которую помещаются данные
- Размер строки
- Спецификатор форматирования
- Дата и время в виде структуры tm

Спецификатор "%T" представляет вывод времени в формате «часы:минуты:секунды», а "%D", - в формате «месяц:день:год». Возможно использование спецификаторов отдельно для получения только необходимых параметров.

Данные, полученные из среды Node-Red, записываем в буфер

```
char str[255]={0};  
  
strcat(str, (char*)message->payload);
```

Для записи в файл нам потребуется функция fprintf()

```
fprintf(file,"Temp: %s Time: %s Date: %s\n",str,time,date);
```

Важным моментом является необходимость закрытия файла, в противном случае данные не сохранятся.

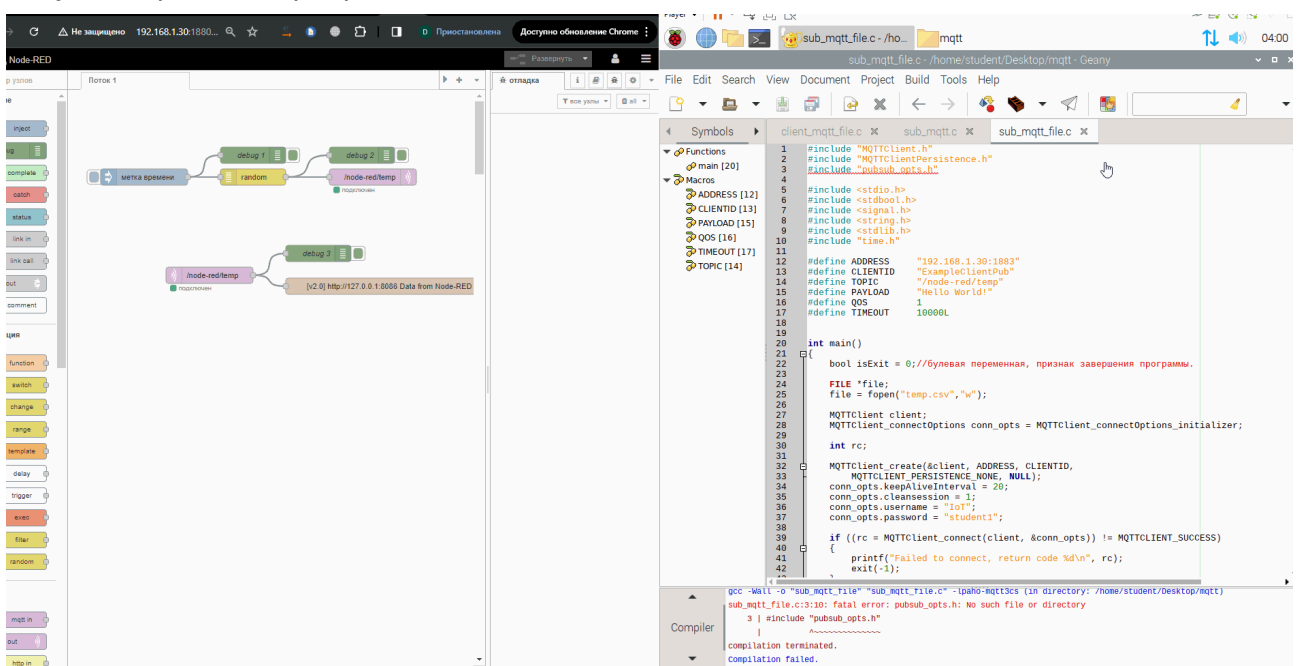
Для решения этой проблемы предусмотрим систему завершения программы. Считаем с клавиатуры знак завершения цикла.

```
char c;  
while((c=getchar())!='\n')  
{  
    if(c=='#')  
    {  
        isExit=1;  
    }  
}
```

При нажатии цикл завершается, мы закрываем файл и отписываемся от издателя.

```
exit:  
fclose(file);  
  
MQTTClient_disconnect(client, 0);  
  
MQTTClient_destroy(&client);  
  
return EXIT_SUCCESS;
```

Результат работы программы:



Решение задания студентами

Тайминг – 10 минут

Студентам предлагается проделать практическое задание №3 самостоятельно.

Курсовая работа

В рамках курсовой работы студентам предложено разработать собственную систему сбора информации для «умного дома» (температура, влажность, загазованность, потребление энергии и т. д – по выбору обучающегося) , построенную на основе протокола обмена MQTT.

Датчик записывает в отдельный файл массив данных, состоящий из даты измерения и величину температуры.

Издатель считывает текстовый файл и отправляет его в среду разработки Node-Red по протоколу обмена MQTT через фиксированный временной интервал.

Подписчик подписан на среду Node-Red и считывает значения датчика. Подписчик записывает полученные данные в файл с указанием времени и даты записи.

Программы издателя и подписчика работают одновременно.

Предполагается, что пользователь может с консоли самостоятельно задавать:

- IP-адрес подключения,
- логин и пароль подключения,
- имена файлов, в которые будут записаны показания датчиков.

В качестве дополнительного задания (*) предусмотрена разработка веб-интерфейса, в котором будут наиболее наглядно и информативно отображены процессы сбора информации с датчиков.

Данный проект является базой в разработке распределенной сети сбора и обработки информации устройств IoT. Эта курсовая работа может стать основой для распределенной системы сбора и анализа информации Raspberry Pi и выделенного сервера, при этом, в перспективе устройства Raspberry Pi могут быть заменены на устройства Arduino. Выполнение данной курсовой работы характеризует обучающегося как специалиста по работе с сетевыми протоколами, облачными вычислениями, веб-технологиями.

Подведем итоги:

- Освоили применение протокола MQTT на практике через рассмотрение конкретных примеров его использования.
- Написали собственное приложение, работающее по принципу «издатель - подписчик»
- Научились рефакторить код приложения для повышения эффективности его работы и улучшения производительности.
- Поставили цели для разработки собственного проекта для портфолио

Формат сдачи и критерии оценки курсовой работы

Критерии проверки домашнего задания:

Выполнено 1 задание - **удовлетворительно**

Выполнено 2 задания - **хорошо**

Выполнено 3 - **отлично**

Формат сдачи курсовой работы:

Студент присылает листинг программы github и скрин ее успешной работы.

Время на проверку преподавателем от 30 до 60 минут

Ответы на вопросы студентов

Тайминг – 5 минут