Рой дронов

Введение

Начнем потихоньку общаться о рое дронов. Сперва попытаемся понять, а что же такое рой. Как гласит одна статья:

“Роевое управление представляет собой подход, изучающий возможности построения системы из совокупности автономных интеллектуальных агентов (роботов) для достижения коллективных целей, которые не могут быть достигнуты отдельным роботом или для которых коллективное выполнение поставленной задачи более эффективно. Основополагающей идеей роевого управления является «роевой интеллект» (РИ, Swarm Intelligence).”

Выделяют два вида роевого управления:

1) Рой без централизованной системы управления;
2) Рой с централизованной системой управления.

В первом случае рой роботов выполняет какую-то общую цель и не контролируется какой-то системой или “лидером”. В этом случае мы имеем многоагентную систему, которая обладает самоорганизующимся поведением и которая, суммарно, должна проявлять некоторое «разумное» поведение.

Во втором случае имеется некий лидер или отдельное устройство, которые принимают решения дальнейшего управления и поведения роя.

Для второго случая можно выделить три стратегии управления:

1) централизованная — дистанционное управление с выделенной базовой станцией, лидер роя назначается из центрального узла;

2) децентрализованная — лидер роя определяется на основе какого-либо алгоритма и не зависит от центральной управляющей станции;
3) смешанная — совмещает в себе преимущества централизованной и децентрализованной стратегий путем выделения лидера роя на основе одного из алгоритмов с передачей прав управления оператору при необходимости.

Мы будем заниматься созданием роя с централизованной системой управления.

1 Входные данные

Исходя из определения роя дронов можно выделить некоторый минимальный функционал того, что должны уметь дроны:

1) Для роевого управления с централизованной системой нужна центральная управляющая станция. Соответственно нужен ноутбук или компьютер с wifi модулем.
2) Для того, чтобы **управляющая станция** могла управлять дронами, эти самый дроны должны иметь возможность общаться с базовой станцией.
3) Для более гибкой реализации на дронах желательно иметь некий вычислительный модуль.

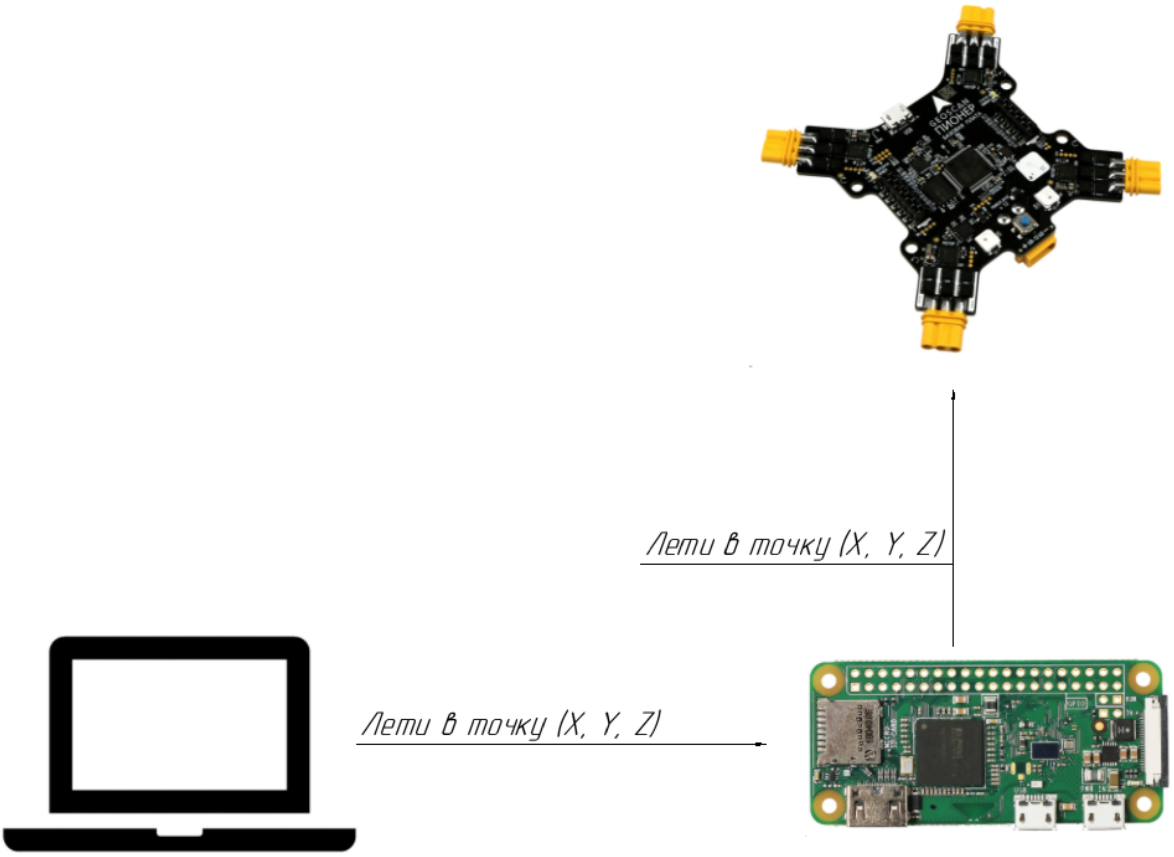
Под 2 и 3 пункты идеально подходит одноплатный компьютер raspberry pi.

Итого для роевого управления имеется: ПК и дрон с raspberry.

2 Анализ задачи

Для анализа задачи представим реализацию одной из основных команд от управляющей станции. Такой командой будет отправка дрона в нужные координаты.

Что мы имеем на первом этапе:



Компьютер отправляет на raspberry каким-то образом команду, которая будет означать отправку коптера в новую точку. Raspberry эту команду принимает и отдает на плату управления для выполнения.

Raspberry и плата управления Пионером связаны физически, поэтому необходим интерфейс для передачи данных между ними. Изучая документацию для Пионера можно обнаружить, что пользователям доступен uart. На этом интерфейсе общения между raspberry и Пионером и остановимся.

Raspberry и компьютер не связаны физически и находятся в некотором удалении друг от друга, соответственно, нужен интерфейс, позволяющий дистанционно общаться. Так как raspberry имеет wifi, то решением будет использование udp сокетов.

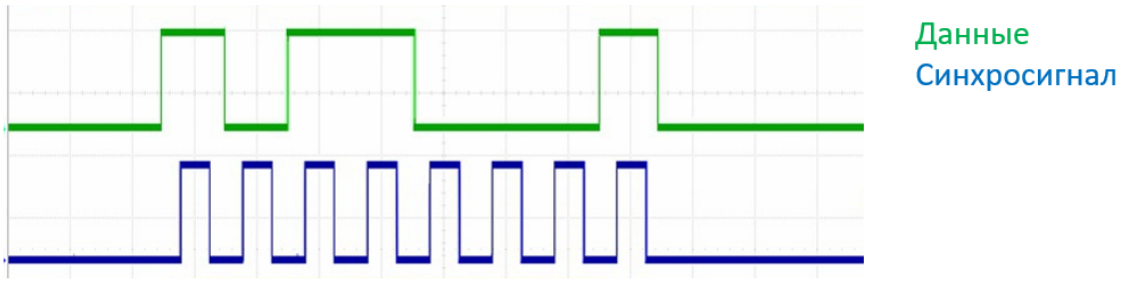
3 UART интерфейс

Начнем решать проблемы по мере их поступления. Сперва нужно разобраться с UART интерфейсом и как им пользоваться для связи общения между raspberry и Пионером.

USART – Универсальный Синхронно-Асинхронный Приемо-Передатчик (УСАПП), который очень часто только асинхронный и поэтому UART.

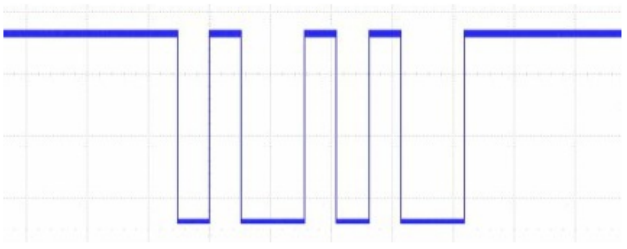
Это интерфейс физического уровня, т.е. он описывает только как передавать биты, без адресации, без защит и т.п.

Рассмотрим сначала **синхронный интерфейс**:



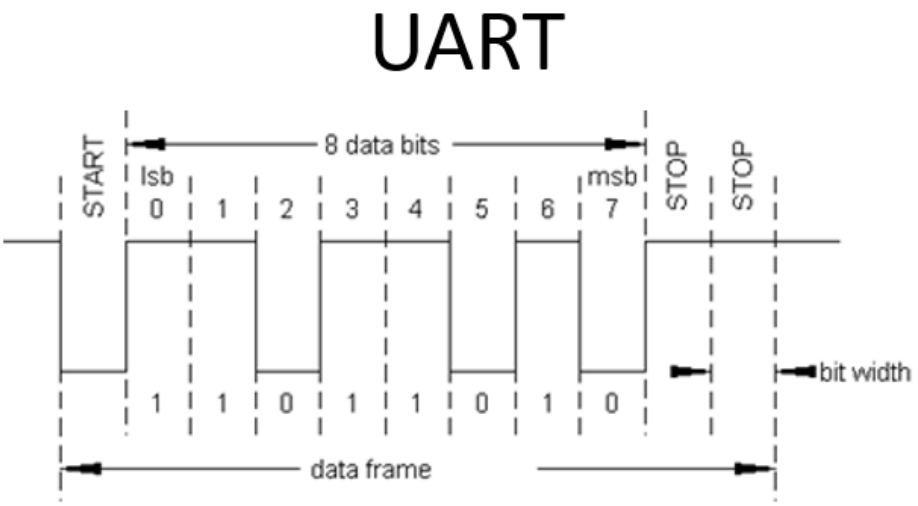
У нас есть два разных провода, по одному идут данные, а по другому синхросигнал, который показывает нам, когда нужно считывать данные. 0 и 1 считываются в момент восходящего фронта синхросигнала. На данной картинке передается байт 0xB1 то есть (1011 0001).

Рассмотрим **асинхронный интерфейс**:



И как тут что-то понять? В какой момент нам считывать 0 или 1, когда начать, когда закончить? Для этого нам нужно знать следующие вещи: когда начать, сколько по времени считывать один бит.

Как выглядит посылка данных в uart интерфейсе:



Когда мы не отправляем никакие данные, то на линии высокий уровень. В пакете 5..9 бит данных, 1 старт бит, 0.5..2 стоп-бита, бит четности.

Какие настройки есть у uart интерфеса:

1) Длина пакета – 5..9 бит данных (обычно 8);
2) Кол-во стоп-битов – 1, 1.5, 2 (обычно 1);
3) Бит четности – есть, нет, всегда 1, всегда 0 (обычно нет);
4) Скорость (бодрейт) в бодах (бит/с) – теоретически любая, обычно – из «стандартного» ряда скоростей (9600, 28800, 57600, 115200...).

Настройки должны быть одинаковыми на приемнике и на передатчике!

3.1 Эксперименты с uart

Сперва установим библиотеку для работы с uart:

pip install pyserial

Импорт библиотеки в проект выполняется с помощью команды: **import serial**

Сперва определимся с командой, которую мы хотим отправить по uart. Пускай это будет команда отправки ковых координат коптеру, например, такая:

CC X Y Z “\n”,

где “CC” - это аббревиатура от “Coordinates Copter”; “X” , “Y”, “Z” - это координаты указанные в метрах (тип данных float); “\n” - символ окончания строки.

Сообщения для отправки будем упаковывать в байты с помощью библиотеки **struct.** Для упаковки и распаковки будет использовать методы **struct.puck()** и **struct.unpuck().** Метод принимает в себя два параметра: ключ, по которому мы производим упаковку и распаковку, и сами данные.

Более подробно о упаковке данных можно изучить тут: https://docs.python.org/3/library/struct.html

Проанализировав таблицу с ключами можем выяснить, что для упаковки нашего сообщения нужен следующий ключ: “>2sfff1c”.

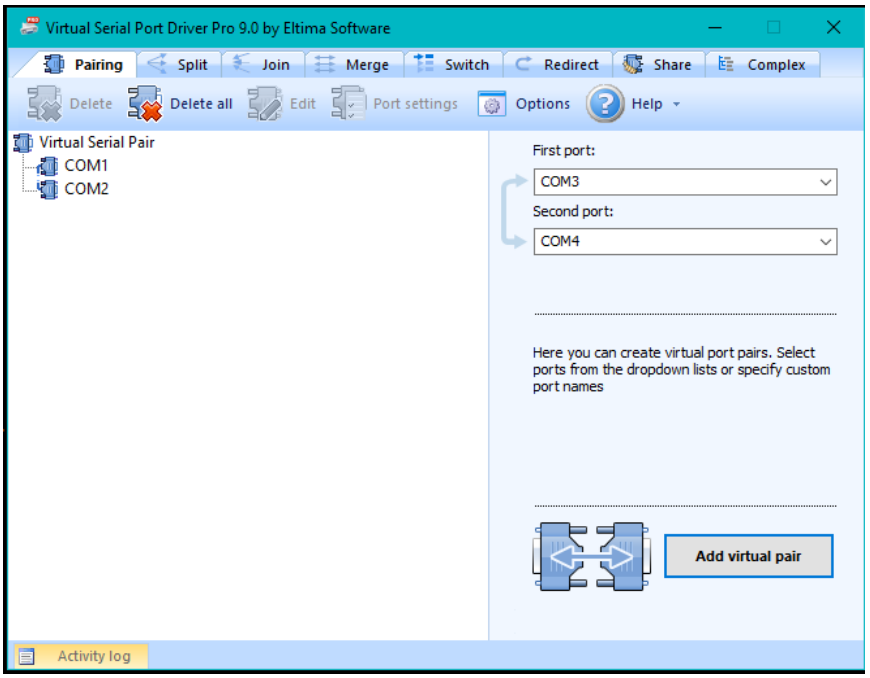
Разберем символы по порядку:

1) “>” - означает, что мы будем использовать стандартный размер типов данных и порядок байт будет идти от старшего байта к младшему (big-endian). Зачем это нужно? Так как в разных системах настройки могут отличаться, то мы сразу явно задаем то, как все упаковывается и распаковывается, чтобы получить одинаковый результат на разных платформах.
2) “2s” - упаковываем два байта. Для упаковки “CC” данную строку сперва нужно представить в байтовом виде (b"str" вернет str в виде байт)
3) “fff” - упаковываем X, Y, Z как данные типа float.
4) “1c” - упаковываем символ окончания строки.

3.2 Реализация

Проверять работу будет на ПК используя виртуальные COM порты. Можно взять usb-uart преобразователь и обойтись без виртуальных портов, но для первых экспериментов удобней все делать на одном только ПК.

Чтобы создать виртуальные порты в своем ПК и связать их друг с другом удобно воспользоваться программой Launch Virtual Serial Port Driver.



Создаем виртуальный порт. В данном случае создана пара COM1 - COM2. В дальнейшем к ним и будет обращаться.

В проекте создадим класс Uart, который будет выполнять подключение, отправку и прием сообщений.

```python
class Uart:
    def __init__(self, port, baud=9600, timeout=1):
        try:
            self.uart = serial.Serial(port, baud, timeout=timeout)
            self.uart.flush()  # очистка юарта
            print("Соединение установлено")
        except:
            print("Ошибка подключения")

    # Принять сообщение
    def accept_message(self):
        if self.uart.in_waiting > 0:
            line = self.uart.readline()
            print(line.decode("utf-8").rstrip())

    # Отправить сообщение
    def send_message(self, message):
        self.uart.write(message)
        pass

    # Сгенерировать
    def create_message_CC(self, X, Y, Z):
        return struct.pack(">2sfff1c", b'CC', X, Y, Z,b"\n")
```

Класс Uart содержит в себе: конструктор и функции отправки, приема и генерации сообщения.

В конструкторе происходит создание uart порта, в который мы передаем следующие настройки: port - порт в который мы будет отправлять данные; timeout - время задержки приема данных. Все остальные параметры оставляем по умолчанию. Создание порта происходит в блоке try\except, это сделано для того, чтобы в случае ошибки программа крашилась, а выполняла блок except.

В функции принятия сообщения сперва выполняется проверка буфера принятых данных и, если буфер не пуст, то мы считываем оттуда данные до того момента, пока не наткнемся на символ завершения строки (“\n”).

Функция отправки сообщения отправляет упакованное сообщение их функции создания сообщения.

4 Socket

Как было сказано выше, необходимо осуществлять общение между raspberry и ПК. Для этих целей можно воспользоваться библиотекой **socket**, которая позволяет создавать клиент\серверные приложения.

Установка библиотеки осуществляется с помощью команды:

pip install sockets

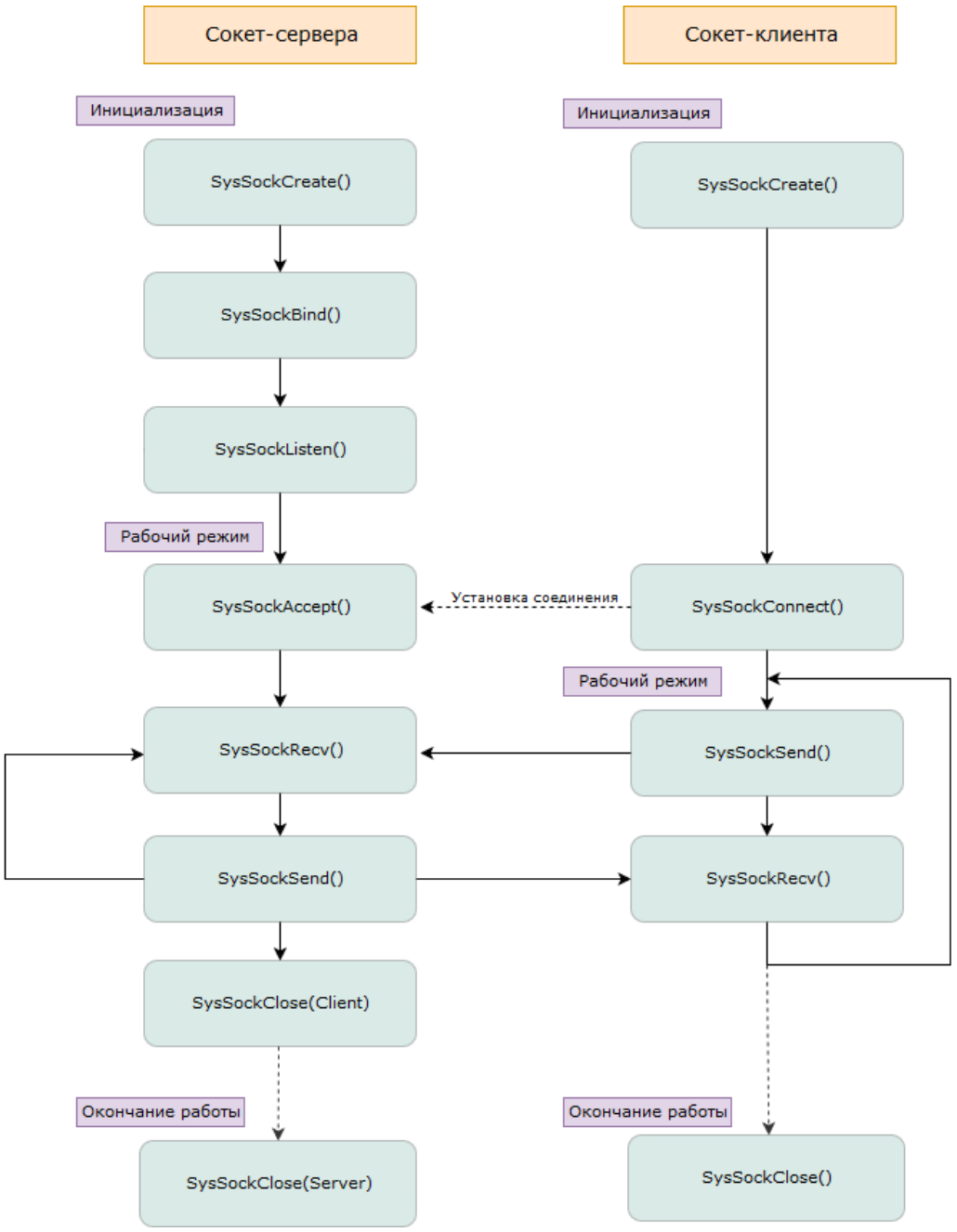
Что же такое сокеты? Сокет - это программный интерфейс для обеспечения информационного обмена между процессами. Существуют клиентские и серверные сокеты. Серверный сокет прослушивает определенный порт, а клиентский подключается к серверу. После того, как было установлено соединение начинается обмен данными.

Библиотека socket позволяет создать два вида сокетов: UDP и TCP. Для начала стоит понять, в чем отличие между ними.

4.1 TCP socket

TCP сокеты используют TCP-соединения, в которых на транспортном уровне обеспечивается надежная доставка данных. TCP протокол отвечает за установление и поддержание соединения, сегментацию, доставку и буферизацию данных, упорядочивание и избавление от дублированных TCP-сегментов данных, контроль ошибок и скорости передачи. Пример работы представлен на рисунке ниже.

Для удобства в качестве функций, указанных на диаграмме, используются функции, из библиотеки socket.



Описание работы сервера. Сперва происходит создание сокета (SockCreate), далее его привязка к конкретному порту и ip адресу (SockBind). После этого сокет ждет к себе подключения клиента и начинается обмен данными. В одном потоке сокет может обработать только одного клиента, потому при подключении нового клиента необходимо создавать новый поток для его обработки.

TCP сокеты гарантируют доставку сообщений и правильный порядок пакетов, а также пересылают пакеты повторно, если

подтверждение о передаче не приходит в течение определенного промежутка времени. Таким образом, использовать TCP сокеты уместно там, где необходима гарантированная доставка данных сетевыми средствами.

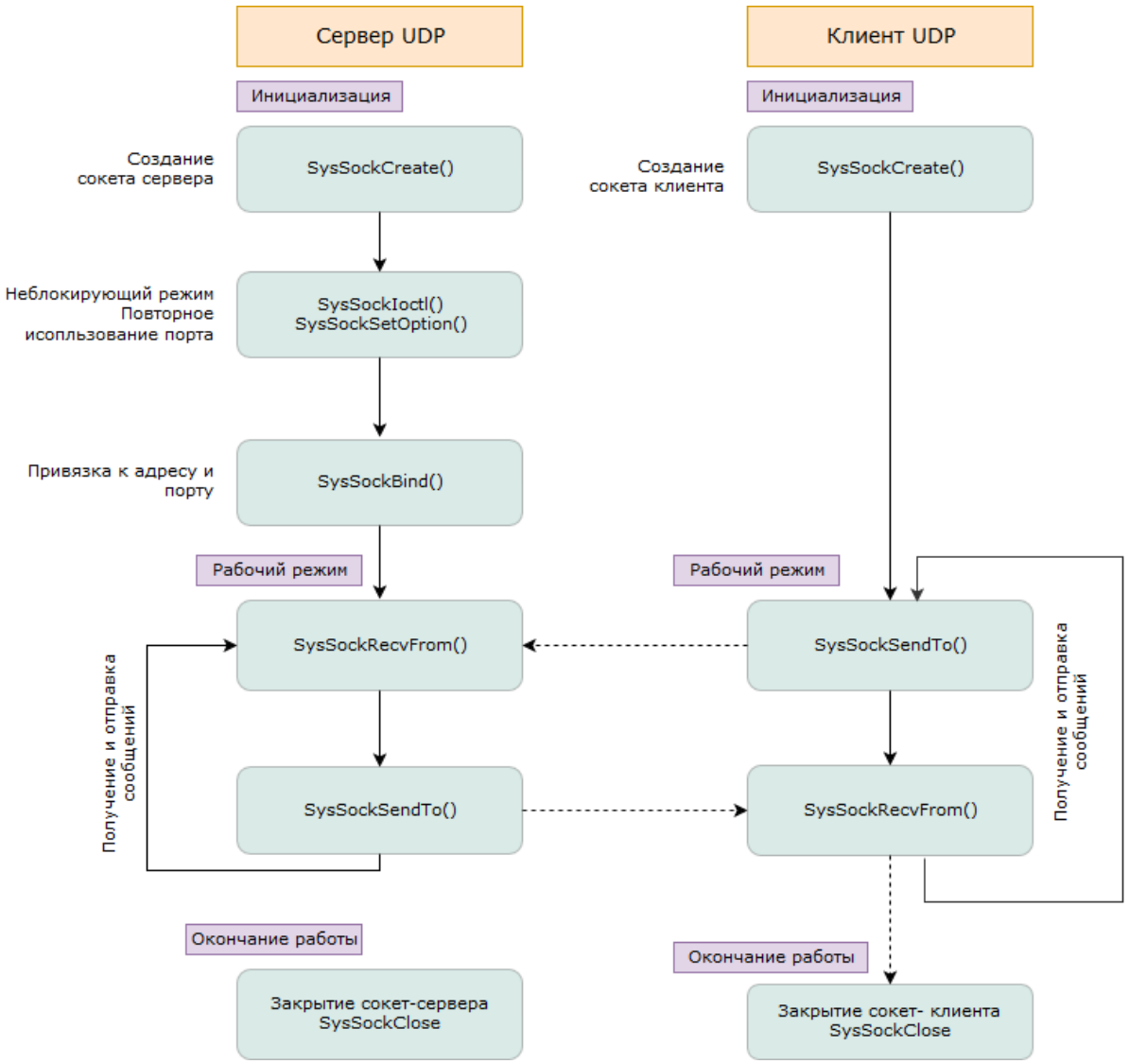
Несмотря на многие преимущества, TCP сокеты имеют и негативные стороны. Например, необходимость поддержания TCP-соединения уменьшает пропускную способность обмена данными в распределенных системах. Также, в системах обмена данными реального времени повторная передача потерянных пакетов может привести к тому, что система получит данные, которые утратили свою актуальность.

4.2 UDP socket

Все перечисленные недостатки TCP сокетов связаны с особенностью TCP-протокола. Если в системе присутствие данных факторов крайне нежелательно, а гарантированность доставки сообщений не является критичным требованием, то в качестве альтернативы TCP сокетов могут использоваться UDP (датаграммные) сокеты.

UDP сокеты устроены проще, чем TCP. В качестве транспортного уровня используется протокол UDP, который не требует установления соединения и подтверждения приема. Информация пересылается в предположении, что принимающая сторона ее ожидает. Датаграммные сокеты не контролирует ничего, кроме целостности полученных датаграмм. Несмотря на это, UDP сокеты нашли свое применение в системах, где на первом месте стоит именно актуальность данных и их быстрая доставка, а не гарантия доставки каждого сообщения.

Пример работы UPD:



Если сравнивать пример работы TCP и UDP сервера, то можно увидеть, что у UDP отсутствует функция подключения клиента. То есть для обработки новых клиентов нет нужды создавать создавать новые потоки и тратить на это вычислительную мощность.

Проанализировав нашу задачу можно понять, что UDP подходит больше, так как позволяет тратить меньше ресурсов на обработку сообщений от большого количества клиентов.