# Лабораторная работа № 1
# Клиент-серверное приложение для передачи сообщений с использованием протоколов TCP и UDP

## 1.1. Цель работы:

Целью первой лабораторной работы является ознакомление студентов с основной структурой клиент-серверного приложения, а также со способами реализации данных приложений с помощью библиотеки Qt.

## 1.2. Задание на лабораторную работу:

Разработать два клиент-серверных приложения, основанных на транспортных протоколах TCP и UDP. Каждое клиент-серверное приложение должно состоять из двух самостоятельных модулей: клиентской части и серверной части.

Клиент-серверное приложение, основанное на транспортном протоколе TCP. Клиент должен инициировать соединение с сервером и отправлять ему текстовые сообщения. Сервер должен принимать сообщения клиента и выводить их на экран, а также иметь возможность отвечать на принятые сообщения.

Клиент-серверное приложение, основанное на транспортном протоколе UDP. Клиент и сервер имеют одинаковое строение, они оба должны иметь возможность отправлять сообщения друг другу и выводить их на экран.

Все приложения должны иметь простейший графический интерфейс.

Для данной лабораторной работы необходимо использовать неблокируемые (асинхронные) сокеты.

## 1.3. Методические указания:

### 1.3.1. Понятие сокет:

Сокеты (sockets) - программный высокоуровневый интерфейс, предоставляющий возможность взаимодействовать с телекоммуникационными протоколами. Сокет представляет собой некую конечную точку коммутации, с помощью которой ваше приложение может обмениваться данными по локальной сети или глобальной сети интернет. Процесс обмена данными с помощью данного интерфейса достаточно прост: приложение записывает данные, которые необходимо отправить, в сокет; отправка данных, их транспортировка осуществляется используемым стеком протоколов и сетевой аппаратурой; при поступлении данных в сокет, приложение считывает пришедшие данные.

### 1.3.2. Обзор сокетов

Сокеты делятся на несколько видов.

В зависимости от используемого транспортного протокола сокеты делятся на потоковые и датаграмные.

Потоковые сокеты – сокеты, использующие, в качестве протокола транспортного уровня, протокол TCP. Данные сокеты отличаются тем, что используют процедуру установления соединения, в результате которой устанавливается некий логический канал. Использование потоковых сокетов гарантирует успешную доставку данных, что обеспечивается механизмом квитирования.

Датаграмные сокеты - сокеты, использующие, в качестве протокола транспортного уровня, протокол UDP. Данные сокеты не используют процедуру установления соединения и не гарантируют успешную доставку данных, однако, в связи с тем, что не происходит передачи служебной информации датаграмные сокеты являются более быстрыми.

Выбор между датаграмными и потоковыми сокетами определяется данными, которые будут передаваться через них. В случае, если необходимо передавать чувствительные к потерям данные (документы, важные сообщения) лучше использовать потоковые сокеты, когда же необходимо передавать чувствительные к задержкам данные (видео, аудиоинформацию в реальном времени) лучше использовать датаграмные сокеты.

В зависимости от процедуры управления сокеты делятся на синхронные и асинхронные.

Синхронные (блокируемые) – сокеты, задерживающие управление на время выполнения операции.

Асинхронные (неблокируемые) – сокеты, возвращающие управления незамедлительно, продолжая работы в фоновом режиме. В данной лабораторной работе будут рассмотрен именно этот тип сокетов.

## 1.4. Особенности работы с сокетами в библиотеке Qt

Библиотека Qt содержит модуль для работы с сетью QtNetwork, который позволяет использовать не только классы для организации транспортного уровня модели OSI (QTcpSocket, QTcpServer, QUdpSocket), но и классы для организации прикладного уровня модели OSI (QHttp, QFtp). Также в модуле QNetwork содержится класс QSslSocket, который позволяет создавать зашифрованные соединения с использованием SSL,TLS.

Для работы с модулем QtNetwork необходимо подключить его к своему проекту. В файле qMake (name_project.pro) дописать следующую строку:

“QT += network”

Также необходимо подключить следующие заголовочные файлы:

#include <QUdpSocket> - класс для UDP сокета
#include <QTcpSocket> - класс для TCP сокета
#include <QTcpServer> - класс для TCP сервера
#include <QHostAddress> - класс для адресации в сети интернет

### 1.4.1. Обзор классов для работы с сокетами

Для работы с асинхронными сокетами необходимо изучить основы библиотеки Qt, а именно механизм сигнал-слот. Данный механизм поддерживается классом QObjekt, поэтому поговорим о нём поподробнее. Диаграмма классов представлена на рисунке1.1

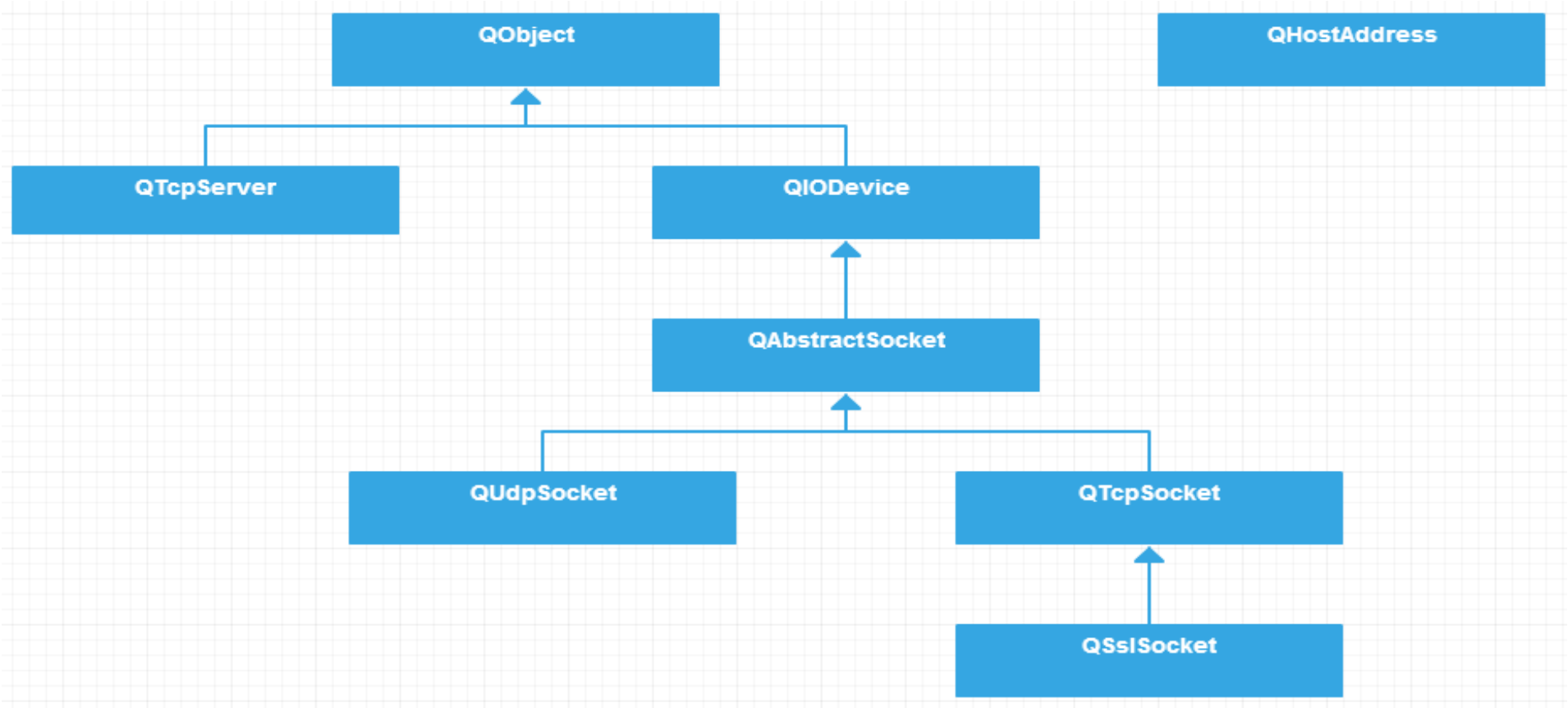


Рисунок 1.1 Диаграмма основных классов модуля QtNetwork

### 1.4.1.1. Класс QObject

QObject - базовый класс для всех объектов Qt.

QObject является сердцем Qt Object Model. Главной особенностью этой модели является очень мощный механизм для связи объектов - механизм сигналов и слотов.

При изменении своего состояния объект испускает асинхронный сигнал. В библиотеку Qt входят множество различных сигналов для каждого объекта. Если пользователю необходимо описать свой собственный сигнал, то в класс, унаследованный от QObject, необходимо добавить:

```
signals:
void mySignal(int numer);
```

Сигналы не требуют реализации, так как предназначены только для передачи информации об изменении состояния (если у сигнала нет параметров) и/или для передачи самих параметров.

Для вызова сигнала:

```
emit mySignal(23);
```

После чего сигнал будет послан в Qt Object Model, для дальнейшей обработки и вызова слота.

Чтобы создать асинхронную реакцию на поступивший сигнал, необходимо:

- создать слот – функция, которая будет выполняться при получении сигнала;
- подключить сигнал к слоту.

В библиотеке Qt реализованы основные слоты, при необходимости пользователь может написать свой собственный слот, для этого в класс, унаследованный от QObject, необходимо добавить:

```
public slots:
void mySlot(int numer);
```

Слоты обладают областью видимости:

public – вызов может происходить из любого объекта;

private - вызов может происходить только из объекта, где слот объявлен;

protected - вызов может происходить не только из объекта, где слот объявлен, но и из объектов родителей и наследников.

Слоты в реализации не отличаются от обычной функции.

Для привязки сигнала к слоту используется функция:

**Таблица 1.1**

| | |
|---|---|
| Функция | QMetaObject::Connection QObject::connect(<br>const QObject * sender,<br>constchar * signal,<br>const QObject * receiver,<br>const char * method,<br>Qt::ConnectionType type = Qt::AutoConnection<br>) |
| Описание функции | Создает соединение сигнала signal от объекта отправителя sender к слоту method объекта получателя receiver. Возвращает дескриптор соединения, который может быть использован, чтобы отсоединить его позже. |
| Описание параметров | sender – объект отправитель сигнала,<br>signal – сигнал объекта отправителя,<br>receiver – объект получатель сигнала,<br>method – слот объекта получателя,<br>type – тип подключения сигнала к слоту. По умолчанию используется Qt :: AutoConnection. |

Пример:

```
connect(myObject,SIGNAL(mySignal(int)), myObject,SLOT(mySlot(int)));
```

Для корректного подключения сигнала к слоту необходимо:

- параметры сигнала совпадают с параметрами слота;
- в качестве параметра используются только зарегистрированные типы (пользовательские типы могут быть зарегистрированы с помощью класса QMetaType).

### 1.4.1.2. Классы для непосредственной работы с сокетами

QTcpSocket - класс для организации потокового сокета. Предоставляет унифицированный интерфейс для работы с потоковыми сокетами. В отличие от Socket API, QTcpSocket не может прослушивать порт, для этого существует специализированный класс QTcpServer.

QTcpServer – класс для организации TCP сервера. Класс позволяет принимать запросы на соединение по протоколу TCP. При поступлении запрос становиться в очередь запросов. При желании сервера установить соединение, создаётся новый QTcpSocket и происходит установление соединения.

QUdpSocket – класс для организации датаграмного сокета. Предоставляет унифицированный интерфейс для работы с датаграмными сокетами.

Подробную работу с этими классами рассмотрим в пункте 1.4.2. и 1.4.3.

QSslSocket – класс для организации потокового сокета, использующий шифрование. QSslSocket аналогичен QTcpSocket. Класс позволяет использовать современные протоколы шифрования такие как SSLv3 и TLSv1_0. Подробную работу с этим классом рассмотрим в лабораторной работе, посвящённой протоколу IMAP.

### 1.4.1.3. QHostAddress

Для работы с IP адресами в библиотека Qt используется класс QHostAddress. Основное предназначение данного класса – хранение, распознавание и преобразование IP адресов. В рамках данной лабораторной работы мы рассмотрим только две функции для хранения адреса.

**Таблица 1.2**

| | |
|---|---|
| Конструктор | QHostAddress::QHostAddress(<br>const QString & address<br>) |
| Описание параметров | address – строка с адресом IPv4 или IPv6 (например “192.168.1.33”) |

**Таблица 1.3**

| | |
|---|---|
| Функция | QString QHostAddress::toString() const |
| Описание функции | Возвращает адрес в удобочитаемом формате |

### 1.4.2. Клиент-серверные приложения с установлением соединения

#### 1.4.2.1. Установка соединения

**Серверная часть**

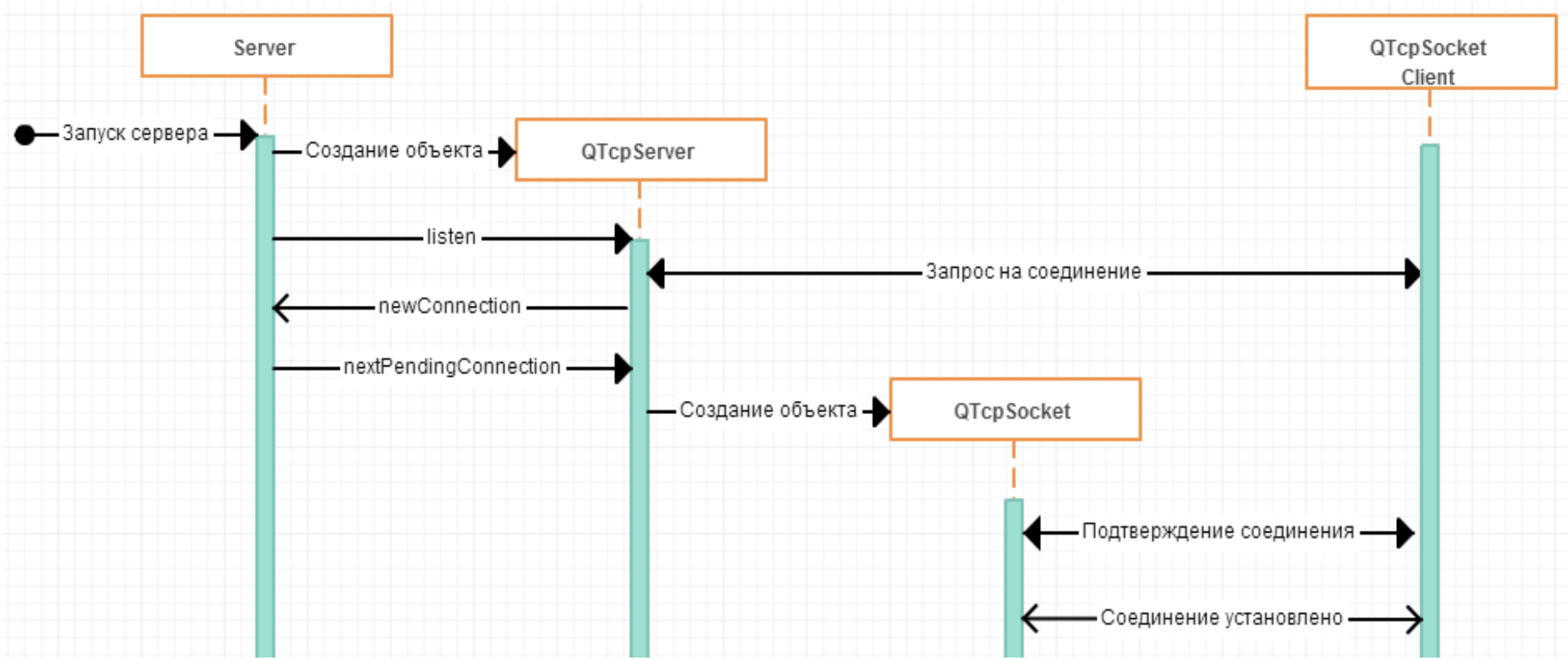


Рисунок 1.2 Диаграмма взаимодействия при установлении соединения (Серверная часть)

На рисунке 1.2 приведена диаграмма взаимодействия при установлении соединения на стороне сервера, рассмотрим каждый шаг:

a) Создание объекта QTcpServer

**Таблица 1.4**

| | |
|---|---|
| Конструктор | QTcpServer::QTcpServer(<br>QObject * parent = 0<br>) |
| Описание параметров | parent – объект владелец QTcpServer. При вызове деструктора объекта владельца так же будут уничтожены все дочерние объекты. В случае если владелец не указан, необходимо вручную вызвать деструктор. |

b) Для запуска сервера необходимо привязать объект к локальному адресу и порту, создать очередь запросов на подключение, перевести сервер в режим прослушивания входящих запросов.

**Таблица 1.5**

| | |
|---|---|
| Функция | bool QTcpServer::listen(<br>const QHostAddress & address = QHostAddress::Any,<br>quint16 port = 0<br>) |
| Описание функции | Привязывает сокет к адресу address и порту port, создаёт очередь запросов на подключение, переводит сервер в режим прослушивания входящих запросов. |
| Описание параметров | address – локальный адрес, в случае если адрес является QHostAddress::Any, сервер будет прослушивать все сетевые интерфейсы.<br>port – локальный порт, в случае если порт = 0, берётся произвольный неиспользуемый системой порт. |

Каждый раз, когда очередной клиент пытается соединиться с сервером, его запрос ставится в очередь, если очередь заполнена, все последующие запросы будут игнорироваться. Для работы с очередью есть несколько функций:

**Таблица 1.6**

| | |
|---|---|
| Функция | void setMaxPendingConnections(<br>int numConnections<br>) |
| Описание функции | Устанавливает максимальный размер очереди |
| Описание параметров | numConnections – максимальный размер очереди |

**Таблица 1.7**

| | |
|---|---|
| Функция | int QTcpServer::maxPendingConnections() const |
| Описание функции | Возвращает максимальный размер очереди |

c) При поступлении запросов на подключение испускается сигнал newConnection (). Для того чтобы отслеживать запросы на подключение, необходимо присоединить сигнал newConnection() к пользовательскому слоту. Чтобы обработать новый запрос необходимо вызвать функцию nextPendingConnection (), которая берёт первый запрос из очереди, создаёт новый сокет для текущего соединения и возвращает указатель на него.

**Таблица 1.8**

| | |
|---|---|
| Функция | QTcpSocket * QTcpServer :: nextPendingConnection () |
| Описание функции | Берёт первый запрос из очереди, создаёт новый сокет для текущего соединения и возвращает указатель на него. |

После этого соединение установлено.

**Клиентская часть**

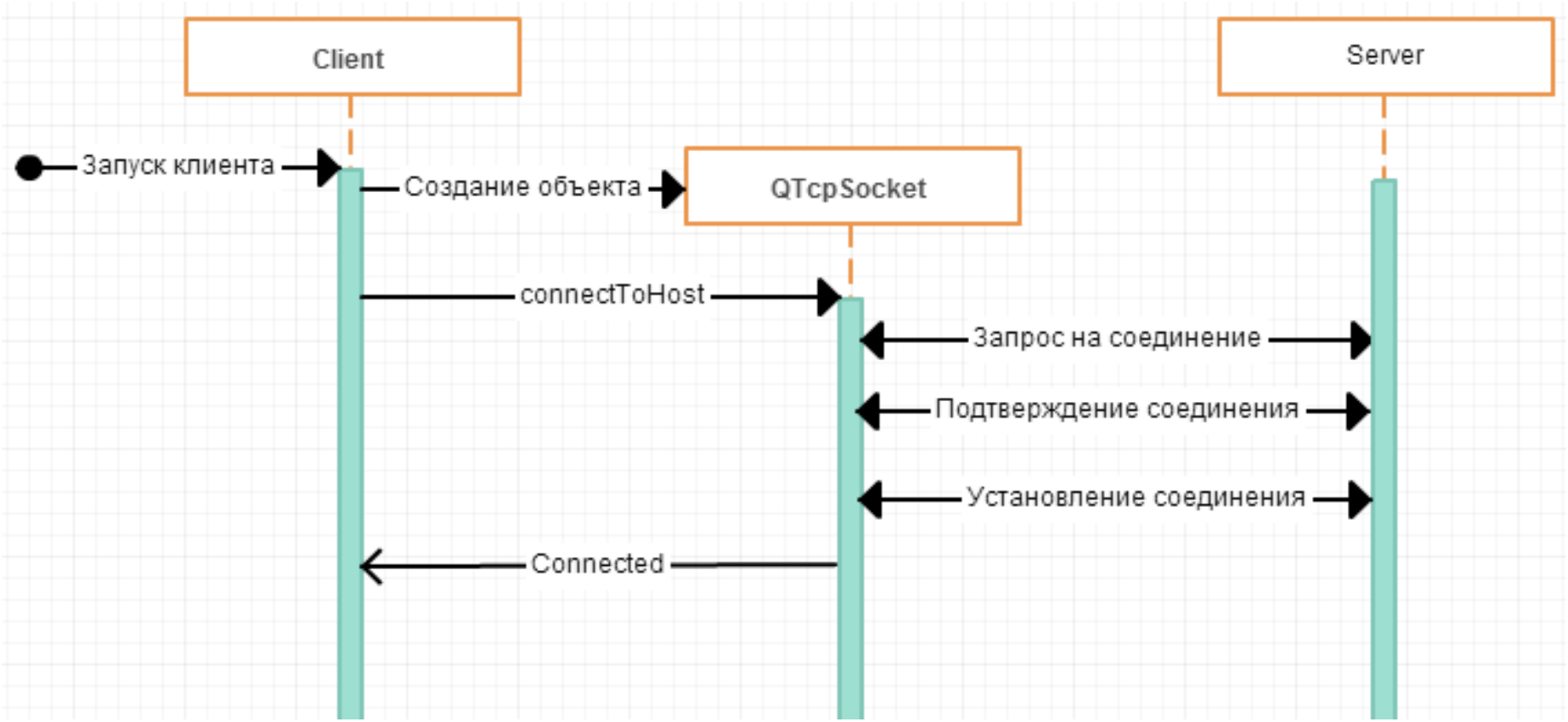


Рисунок 1.3 Диаграмма взаимодействия при установлении соединения (Клиентская часть)

На рисунке 1.3 приведена диаграмма взаимодействия при установлении соединения на стороне клиента, рассмотрим каждый шаг.

В Qt в отличии от SOCKET API не нужно вручную привязывать сокет к определённому локальному адресу или порту, так как до вызова функции connectToHost() сокет не создаётся. При вызове данной функции начинается проверка адреса по всем сетевым интерфейсам, если адрес доступен испускается сигнал hostFound(). При получении сигнала hostFound() создаётся сокет на подходящем сетевом интерфейсе, использующий произвольный свободный порт, и происходит попытка соединения. Если соединение удалось, то посылается сигнал Connected().

**Таблица 1.9**

| | |
|---|---|
| Функция | void QAbstractSocket::connectToHost(<br>const QString & hostName,<br>quint16 port,<br>OpenMode openMode = ReadWrite,<br>NetworkLayerProtocol protocol = AnyIPProtocol<br>) |
| Описание функции | Проверяет доступность адреса hostname и порта port, создаёт системный сокет на подходящем сетевом интерфейсе, используя произвольный незадействованный порт и пытается установить соединение. |
| Описание параметров | hostName – адрес сервера в сети<br>port – порт сервера<br>openMode – режим, в котором будет происходить работа с сокетом. По умолчанию - режим для чтения и записи (ReadWrite)<br>protocol – протокол сетевого уровня модели OSI. По умолчанию - протокол IPv4 или IPv6(AnyIPProtocol) |

### 1.4.2.2. Обмен данными

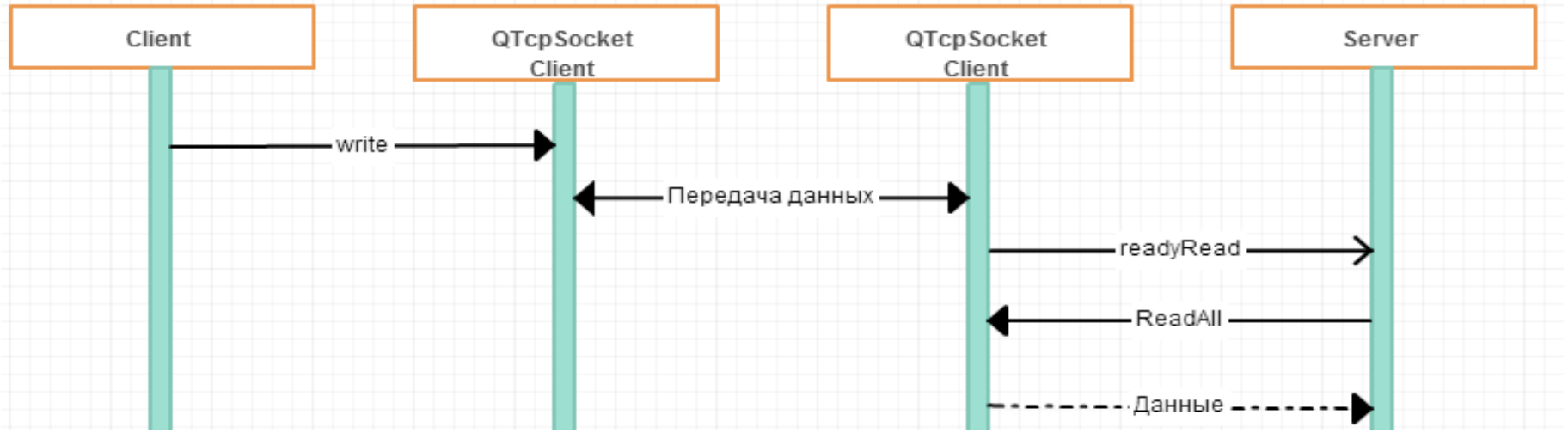


Рисунок 1.4 Диаграмма взаимодействия при передачи данных

На рисунке 1.4 приведена диаграмма взаимодействия при обмене данными, рассмотрим каждый шаг.

Для отправки данных используется функция:

**Таблица 1.10**

| | |
|---|---|
| Функция | qint64 QIODevice::write(<br>const QByteArray & byteArray<br>) |
| Описание функции | Записывает byteArray в буфер для отправки. Возвращает количество байт, которые были записаны в буфер. В случае ошибки возвращает -1. |

После записи данных в буфер для отправки, данные будут поделены на пакеты (если это необходимо) и отправлены получателю.

При получении данных испускается сигнал void QIODevice::readyRead(). Для получения данных из буфера используется функция:

**Таблица 1.11**

| | |
|---|---|
| Функция | QByteArray QIODevice :: ReadAll () |
| Описание функции | Читает данные из буфера, записывает их в массив байт и возвращает его. В случае ошибки возвращает пустой массив. |

Размер буфера в Qt не ограничен.

### 1.4.2.3. Закрытие соединения

Для корректного завершения работы с сокетами используется функция:

**Таблица 1.12**

| | |
|---|---|
| Функция | QAbstractSocket :: disconnectFromHost () |
| Описание функции | Закрывает сокет, освобождает ресурсы. В случае если есть данные, ожидающие записи в буфер, закрытие сокета будет произведено после отправки всех данных. |

Для корректного завершения работы с сервером используется функция:

**Таблица 1.13**

| | |
|---|---|
| Функция | void QTcpServer::close() |
| Описание функции | Сервер перестаёт слушать порт |

В случае разрыва соединения происходит закрытие системного сокета и освобождение ресурсов.

### 1.4.2.4. Обработка ошибок

Для обработки ошибок используется функция:

**Таблица 1.14**

| | |
|---|---|
| Функция | QString QTcpServer::errorString() const |
| Описание функции | Возвращает удобочитаемое описание последней ошибки |

Так же для асинхронной обработки ошибок, при возникновении ошибки испускается сигнал error(QAbstractSocket::SocketError).

### 1.4.3. Клиент-серверные приложения без установления соединения

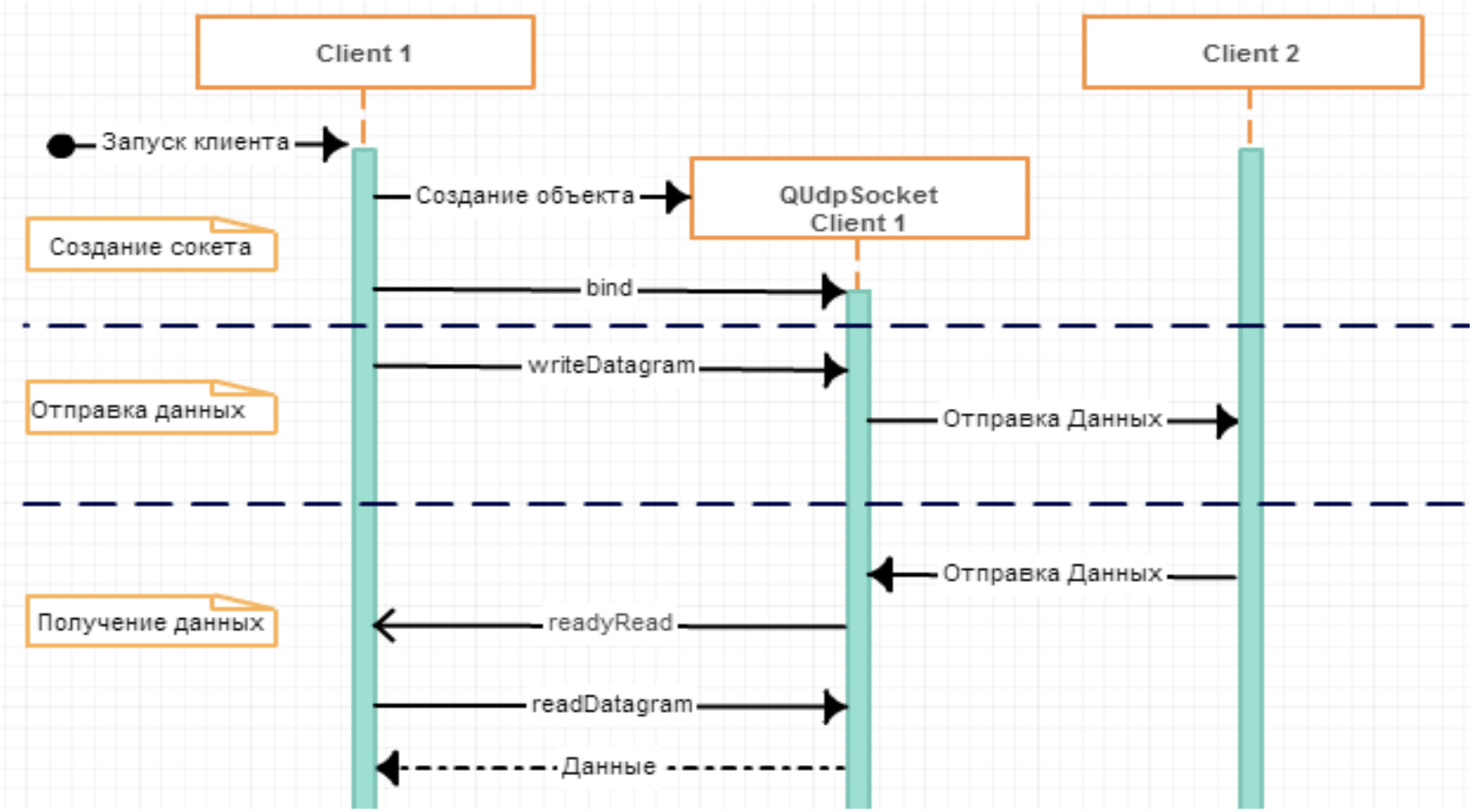


Рисунок 1.5 Диаграмма взаимодействия датаграмных сокетов.

При использовании датаграмных сокетов, не происходит процедуры установления соединения, поэтому их использование гораздо проще.

На рисунке 1.5 приведена диаграмма взаимодействия, при использовании датаграмных сокетов, рассмотрим каждый шаг:

a) Создание объекта QUdpSocket

**Таблица 1.15**

| | |
|---|---|
| Конструктор | QUdpSocket::QUdpSocket(<br>QObject * parent = 0<br>) |
| Описание параметров | parent – объект владелец QUdpSocket. При вызове деструктора объекта владельца так же будут уничтожены все дочерние объекты. В случае если владелец не указан, необходимо вручную вызвать деструктор. |

b) Создание сокета, привязка к локальному адресу и порту

**Таблица 1.16**

| | |
|---|---|
| Функция | bool QAbstractSocket::bind(<br>const QHostAddress & address,<br>quint16 port = 0,<br>BindMode mode = DefaultForPlatform<br>) |
| Описание функции | Создаёт сокет, связывая его с адресов address и портом porr. |
| Описание параметров | address – локальный адресс<br>port – локальный порт, если порт = 0, то используется произвольный не использованный порт.<br>mode – модель привязки сокета к локальному адресу, по умолчанию – стандартный для платформы(DefaultForPlatform) |

c) Отправка сообщений осуществляется с помощью функции:

**Таблица 1.17**

| | |
|---|---|
| Функция | qint64 QUdpSocket::writeDatagram(<br>const QByteArray & datagram,<br>const QHostAddress & host,<br>quint16 port<br>) |
| Описание функции | Отправляет данные datagram, на адрес host и порт potr. В случае, если размер датаграммы слишком большой или произошла ошибка возвращает -1 |
| Описание параметров | datagram – массив байт для отправки<br>host – адрес приёмного устройства<br>port – порт приёмного устройства |

При поступлении датаграммы испускается сигнал void QIODevice::readyRead ().

d) Для получения датаграммы необходимо воспользоваться функцией:

**Таблица 1.18**

| | |
|---|---|
| Функция | qint64 QUdpSocket::readDatagram(<br>char * data,<br>qint64 maxSize,<br>QHostAddress * address= 0,<br>quint16 * port = 0<br>) |
| Описание функции | Читает maxSize байт из буфера, сохраняет их в data, если указаны address и port, то сохраняет адрес отправителя в них. Возвращает количество прочитанных байт, если произошла ошибка, то возвращает -1. |
| Описание параметров | data – массив байт для приёма<br>maxSize – размер массива байт<br>address – адрес отправителя<br>port – порт отправителя |

e) Для закрытия сокета используется функция:

**Таблица 1.19**

| | |
|---|---|
| Функция | void QIODevice::close() |
| Описание функции | Закрывает сокет |

## 1.5. Контрольные вопросы и задания.

1) Описать процесс установления соединения на стороне клиента.
2) Описать процесс установления соединения на стороне сервера.
3) Объяснить различия между обменом пакетами в ориентированных на соединение протоколах и не ориентированных на соединение протоколах.
4) Объяснить различия блокируемых и неблокируемых сокетов.
5) Объяснить, как реализуются асинхронные сокеты, в выбранной вами библиотеке.