ВМСиС

Лекция 8
Разработка сетевых приложений

Разработка приложений с т.з. OSI

Application

HTTP, FTP, SMTP, RDP и другие

Presentation

Session

Пользовательский протокол использующий транспортный и\или сетевой уровень

Приложения использующие стандартные протоколы уровня Application

Применяется в тех случаях, когда необходимо предоставить максимально стандартизированную среду передачи данных и нет потребности в особых решениях

- Приложения использующие HTTP-based API
- Веб-браузеры
- Почтовые клиенты

Приложения использующие пользовательские протоколы

Применяется в тех случаях, когда отсутствует возможность или потребность в использовании стандартных протоколах

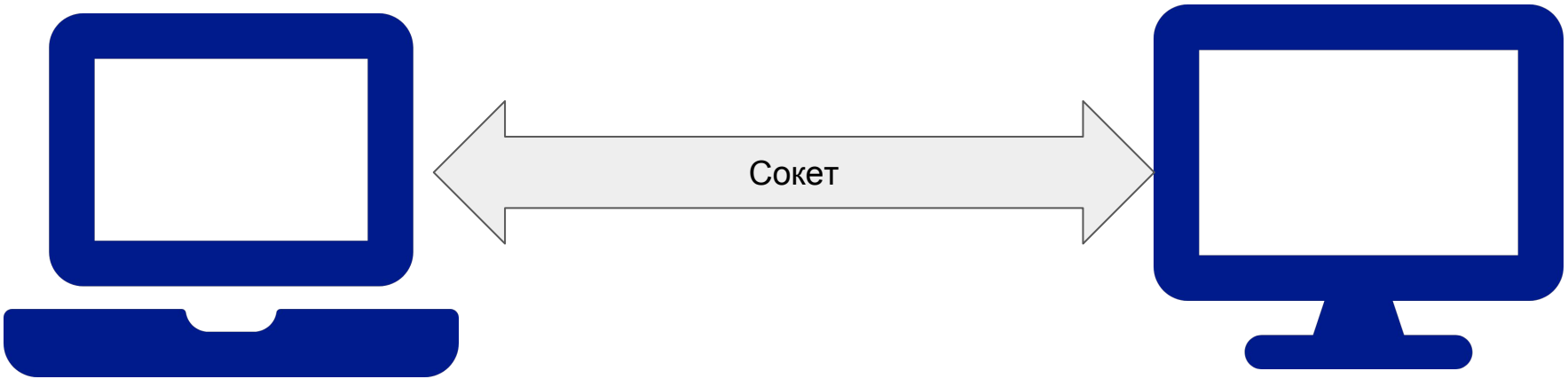
Преимущества:

- Возможность тонкой настройки под потребности
- Отсутствие балластного оверхеда
- Возможно добиться более высокой скорости работы

Сокеты (sockets)

Сокеты - это программный интерфейс позволяющий соединить две программы запущенные как на локальной машине, так и на удаленных.

Сокеты предоставляют программный доступ к протоколам TCP\UDP



Общий принцип работы UDP сокетов

Клиент	Сервер
	Активирует слушающий сокет по заданному порту (например 443)
Клиент отправляет данные	
	Сервер принимает данные

Типы TCP сокетов

- Серверный (слушающий) сокет
  - Ожидает подключения извне

- Клиентский сокет
  - Подключается к серверному сокету и устанавливает соединение

Общий принцип работы TCP сокетов

Клиент	Сервер
	Активирует слушающий сокет по заданному порту (например 443)
Создает сокет и подключает его по адресу и порту сервера	
	Принимает подключение и создает новый сокет, который будет работать с данным клиентом
Начинает обмен данными с сервером	
	Слушающий сокет продолжает работать и ожидать следующего клиента

Основные функции

Общие

- Send - отправить данные
- Receive - принять данные
- Close - прервать соединение

Серверный сокет

- Bind - связать сокет с IP-адресом и портом
- Listen - начать процесс ожидания подключения
- Accept - принять запрос на установку соединения

Клиентский

- Connect - установить соединение по указанному адресу и порту

Блокирующие операции

Операция которая приостанавливает выполнение своего потока называется блокирующей

К блокирующим относятся:

- Send
- Receive
- Connect
- Listen
- Accept

Если вызвать блокирующую операцию, то программа “зависнет” до тех пор, пока операция не будет завершена

Многопоточное программирование

Современные ОС позволяют разделять приложение на несколько одновременно выполняющихся потоков. Это позволяет выполнять блокирующие операции отдельно от пользовательского интерфейса

А что C#?

Для работы с сокетами C# предоставляет следующие обертки:

- **TcpListener** - серверный TCP сокет
- **TcpClient** - Клиентский TCP сокет
- **UdpClient** - UDP сокет, может быть как сервером так и клиентом

Для создания новых потоков выполнения используется класс **Thread**

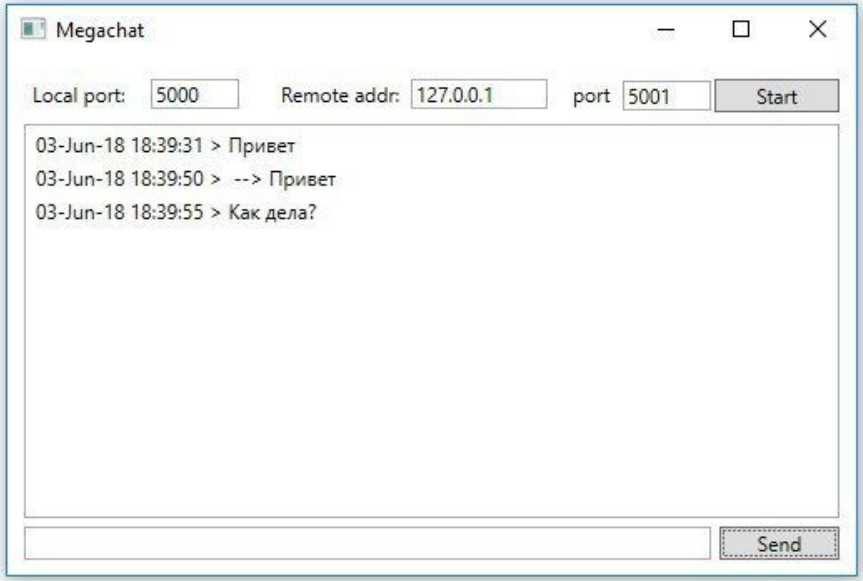
Пример использования UdpClient. Сервер

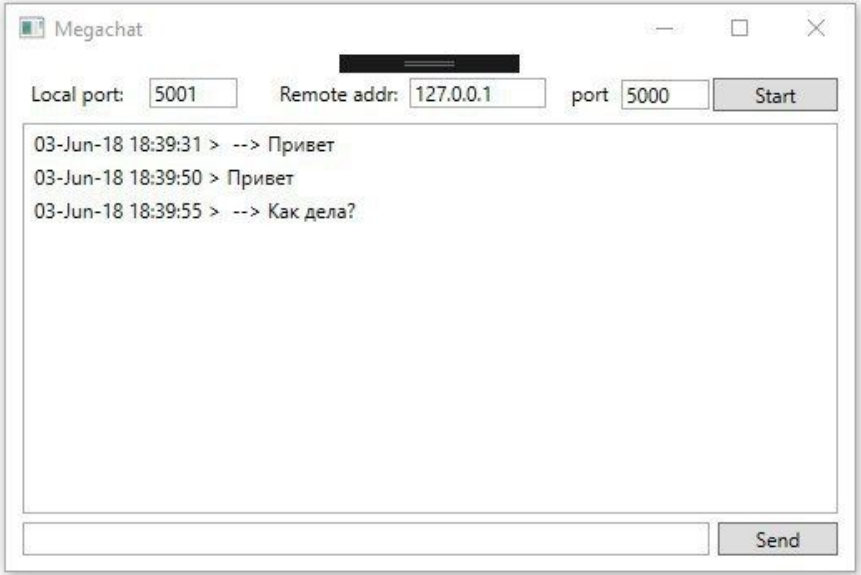
```
UdpClient udpServer = new UdpClient(11000);
while (true) {
    var remoteEP = new IPEndPoint(IPAddress.Any, 11000);
    var data = udpServer.Receive(ref remoteEP);
    Console.Write("receive data from " + remoteEP.ToString());
    udpServer.Send(new byte[] { 1 }, 1, remoteEP);
}
```

Пример использования UdpClient. Сервер

```
var client = new UdpClient();

IPEndPoint ep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 11000);

client.Connect(ep); // send data

client.Send(new byte[] { 1, 2, 3, 4, 5 }, 5); // then receive data

var receivedData = client.Receive(ref ep);

Console.Write("receive data from " + ep.ToString());

Console.Read();
```

Пример приложения обмена сообщениями





```
private void pbStart_Click(object sender, RoutedEventArgs e)
{
    // Получаем номер порта который будет слушать наш клиент
    int port = int.Parse(tbLocalPort.Text);
    // Запускаем метод Receiver в отдельном потоке
    Thread tRec = new Thread(() => Receiver(port));
    tRec.Start();
}
```

```
private void Receiver(int port) {
    receivingUdpClient = new UdpClient(port);
    IPEndPoint remoteIpEndPoint = null;
    try     {
        while (true) {
            // Ожидание дейтаграммы
            byte[] receiveBytes = receivingUdpClient.Receive(ref remoteIpEndPoint);
            // Преобразуем байтовые данные в строку
            string returnData = Encoding.UTF8.GetString(receiveBytes);
            // Выводим данные из нашего потока в основной
            lbLog.Dispatcher.BeginInvoke(new Action( () => addMessage(" --> " + returnData)));
        }
    }
    catch (Exception ex)
    {
        // В случае ошибки завершаем поток и выводим сообщение в лог чата
        lbLog.Dispatcher.BeginInvoke(new Action(() =>
            addMessage("Возникло исключение: " + ex.ToString() + "\n  " + ex.Message))
        );
    }
}
```

```
private void pbSend_Click(object sender, RoutedEventArgs e){
    // Создаем сокет для отправки данных
    UdpClient other = new UdpClient();
    // Создаем endPoint по информации об удаленном хосте
    IPEndPoint endPoint = new IPEndPoint(IPAddress.Parse(tbRemoteAddr.Text),
                                         int.Parse(tbRemotePort.Text));

    // Преобразуем данные в массив байтов, используя кодировку UTF-8
    byte[] bytes = Encoding.UTF8.GetBytes(tbMessage.Text);
    // Отправляем данные
    other.Send(bytes, bytes.Length, endPoint);
    // выводим наше сообщение в лог
    addMessage(tbMessage.Text);
    // очищаем поле ввода
    tbMessage.Clear();
}
```