**Лабораторная работа 1.**

**Разработка программных модулей**

**Элементы сетевого программирования**

Цель: Ознакомиться с теорией и практикой создания элементов сетевого программирования в среде .Net.

**Теория**

**Часть 1. Сетевые понятия и протоколы**

**Физическая модель**

По существу сеть представляет собой группу компьютеров или устройств, соединенных каналами связи (communication links). В терминах сетевой обработки все компьютеры и устройства (принтеры, маршрутизаторы, коммутаторы и т. д.), присоединенные к сети, называются узлами (nodes). Узлы соединены каналами, которые могут быть кабелями или беспроводными соединениями (инфракрасными или радиосигналами) и могут взаимодействовать с другими узлами, передавая сообщения через сеть.

Сети различаются в зависимости от размера:

**Локальная сеть (LAN, или Local Area Network)**

Соединяет узлы, расположенные на ограниченном участке. Этот участок может соответствовать расположению крупной компании или быть совсем небольшим, на котором объединяются компьютеры в жилой квартире. Наиболее часто используемая технология локальных сетей — сеть Ethernet.

**Глобальная сеть (WAN, или Wide Area Network)**

Может объединять несколько участков локальных сетей. Известны такие технологии глобальных сетей, как Frame Relays (с ретрансляцией кадров), линии Tl, Integrated Services Digital Network, или ISDN (цифровая сеть с интегрированными услугами), Х.25 и Asynchronous Transfer Monitor, или ATM (асинхронный режим передачи). Далее мы подробнее рассмотрим средства соединения с глобальной сетью.

**Региональная сеть (MAN, или Metropolitan Area Network)**

Очень похожа на глобальную сеть в том, что она тоже объединяет несколько локальных сетей. Однако региональная сеть сужает территорию сети до города или небольшого района. В региональной сети используются высокоскоростные сети, соединяющие локальные сети школ, правительственных учреждений, компаний и т. д. с помощью таких быстрых соединений, как волоконная оптика.

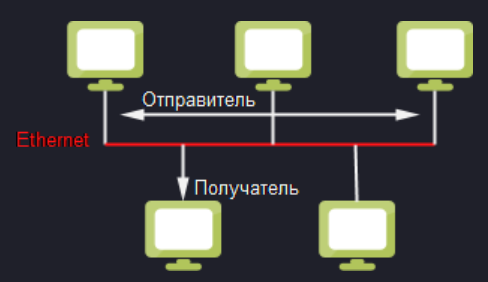
**Протокол Ethernet**

Чтобы лучше понять, как работают физические сети, мы рассмотрим Ethernet, наиболее распространенный протокол для локальных сетей. Из всех устройств, подключенных к локальным сетям, 90% используют протокол Ethernet, который первоначально в 1972 г. разработали компании Xerox, Digital Equipment и Intel. В 1980 г. стандарт IEEE 802.3 CSMA/CD определил протокол Ethernet со скоростью передачи 10 Мбит/сек.

В настоящее время сети Ethernet могут поддерживать линии со скоростями 100 Мбит/сек и 1 Гбит/сек. С Ethernet могут применяться многие кабельные технологии. С помощью стандартизованной системы имен указывают скорость сети Ethernet и свойства кабельной технологии. Эти имена начинаются с числа, означающего максимальную скорость передачи данных, за которым следует слово, указывающее поддерживаемую технологию передачи, и наконец число, определяющее максимальное расстояние между узлами. Например, 10Base2 обозначает сеть Ethernet, функционирующую со скоростью 10 Мбит/сек, использующую узкополосную передачу по кабелям с максимальной длиной 200 м.

**CSMA/CD**

Интернет — это сеть CSMA/CD (множественный доступ с контролем несущей и обнаружением конфликтов (коллизий)). Несколько устройств подсоединены к одной сети, и все они могут обращаться к ней одновременно. Когда посылается сообщение, оно передается по всей сети, как показано на следующем рисунке. Получатель определяется уникальным адресом, и только этот узел читает сообщение, остальные его игнорируют:



В такой схеме заложена потенциальная проблема, поскольку одновременно могут попытаться отправить сообщение не один, а несколько узлов, что приводит к искажению пакетов. Используемое в Ethernet решение состоит в том, что каждый узел прослушивает сеть и, таким образом, знает, есть ли в данный момент трафик в сети. Узел может начать отправку данных, если по сети еще не посланы другие данные. Короче говоря, это часть CSMA (множественный доступ с контролем несущей) протокола CSMA/CD.

Однако по-прежнему остается возможность, что два узла, проверив, что сеть пока не используется, начнут отправку пакета точно в одно время по одному сетевому кабелю. Это приведет к возникновению конфликтов между двумя пакетами, и в результате данные будут искажены. Оба отправителя узнают об искажении пакета, поскольку при посылке данных они продолжают слушать сеть и обнаруживают конфликт.

В протоколе CSMA/CD это часть CD (обнаружение конфликтов). Тогда оба узла немедленно останавливают передачу и ждут в течение случайного промежутка времени, по истечении которого они снова проверяют состояние сети, чтобы выяснить, свободна ли она и можно ли отправить пакет повторно.

Для уникальной идентификации каждый узел локальной сети использует адрес управления доступом к среде (Media Access Control, MAC). Этот адрес определен сетевой интерфейсной платой. Сетевой пакет отправляется по сети, и, если сетевая плата определяет, что пакет не предназначен для ее хоста, она игнорирует пакет и передает его дальше. Если же пакет предназначен для этой платы, она все равно передает его дальше, но на этот раз отмечает в нем, что он получен. Пакет продолжает свой путь по сети, пока не вернется к отправителю, который убеждается, что намеченный адресат получил данные.

**Другие протоколы**

Компания IBM разработала сеть Token Ring (IEEE 802.5), узлы которой соединены в кольцо. В технологии Ethernet любой узел может отправить сообщение, если в сети нет трафика. В сети Token Ring каждый узел имеет гарантированный доступ к сети в предопределенном порядке. По кольцу сети циркулирует маркер, и только узел, владеющий маркером, может отправить сообщение. В настоящее время Ethernet постепенно вытесняет технологию Token Ring как более дорогую и сложную в реализации.

**AppleTalk** — это протокол локальной сети, разработанный компанией Apple, был очень популярным в США.

Асинхронный режим передачи (Asynchronous Transfer Mode, ATM) — еще один протокол, который можно встретить в локальных сетях. Он поддерживает быструю коммутацию сети и имеет гарантированное "качество обслуживания" (Quality of Service, QoS), но, поскольку цена сетевых плат ATM очень высока, ATM применяется лишь в узкоспециализированном секторе рынка локальных сетей. ATM используется лишь для сетей LAN, требующих чрезвычайно высокой производительности, например, для передачи между больницами таких медицинских изображений, как рентгеновские снимки. В магистрали, управляющей глобальными сетями, ATM играет более важную роль.

**Сетевая интерфейсная плата**

Сетевая интерфейсная плата (Network Interface Card, или NIC) — это адаптерная плата, используемая для соединения устройства с локальной сетью. Она позволяет отправлять сообщения в сеть и получать сообщения из сети. Сетевая плата имеет уникальный МАС-адрес, обеспечивающий уникальную идентификацию для каждого устройства.

**МАС-адрес** — это 6-байтное шестнадцатеричное число, уникально назначенное сетевой Ethernet-плате. Этот адрес может быть изменен сетевым драйвером динамически (как, например, в сетях DECnet, разработанной компанией Digital Equipment), но обычно МАС-адрес не изменяется.

Вы можете узнать МАС-адрес машины Windows с помощью утилиты командной строки ipconfig, запустив ее в окне командной строки с установленным переключателем /all.

**Модель OSI**

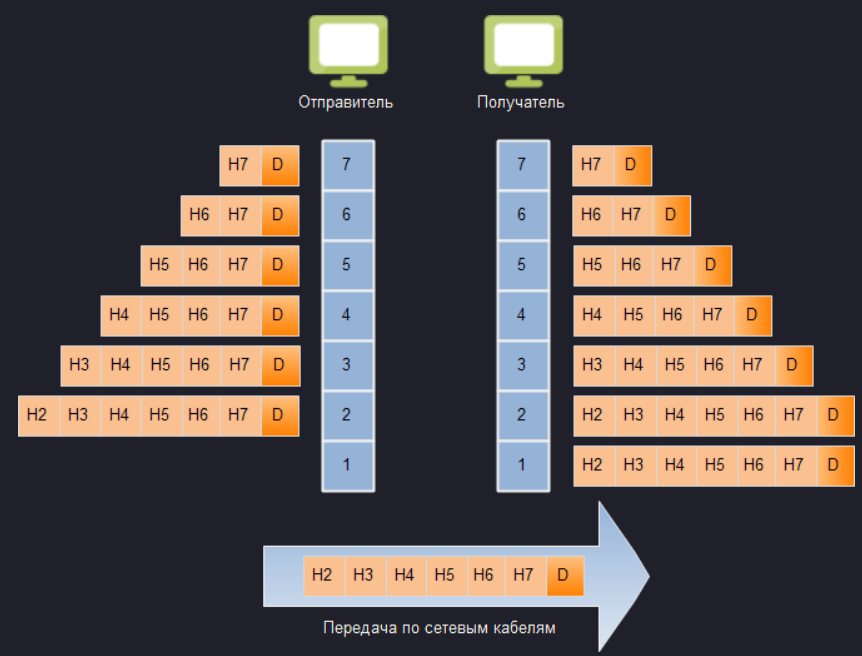
В целях формализации процесса взаимодействия открытых систем OSI (Open System Interconnection) Международная организация по стандартизации (International Organization for Standardization, ISO) разработала для стандартизированной сети модель, которая заменила бы TCP/IP, DECNet и другие протоколы как основной сетевой протокол, использующийся в Интернете. Однако из-за сложности протокола OSI созданы и внедрены в эксплуатацию лишь немногие реализации. TCP/IP был гораздо проще, и поэтому он широко используется и сейчас. Но многие новые идеи протокола OSI можно обнаружить в IPv6, следующей версии IP.

Хотя протокол OSI не получил широкого распространения, семиуровневая модель OSI имела огромный успех, и она теперь используется как справочная модель, чтобы описать различные сетевые протоколы и их функциональные возможности.

Уровни модели OSI выделяют основные задачи, которые должны выполняться сетевыми протоколами, и описывают взаимодействие сетевых приложений. У каждого уровня есть особое назначение, и каждый уровень связан с уровнями, находящимися непосредственно выше и ниже его. OSI определяет следующие семь уровней:

* Прикладной уровень определяет для пользовательских приложений программный интерфейс с сетью.
* Представительский уровень отвечает за кодирование данных, полученных от прикладного уровня, в представление, готовое к передаче по сети, и наоборот.
* Сеансовый уровень создает виртуальное соединение между приложениями.
* Транспортный уровень делает возможным надежный обмен данными.
* Сетевой уровень позволяет обращаться к узлам локальной сети, используя логическую адресацию.
* Канальный уровень получает доступ к физической сети через физические адреса.
* Физический уровень включает соединители, кабели и т. д.

На следующем рисунке показана связь между двумя машинами. Здесь можно увидеть, как данные опускаются по стеку протоколов (protocol stack) на отправителе и поднимаются по нему на получателе. Сообщение, отправленное приложением первой машины, содержащей букву D. Прикладной уровень (уровень 7) присоединяет к сообщению заголовок (названный на рисунке Н7) и передает сообщение представительскому уровню (уровень 6), который сначала добавляет к сообщению Н6, а затем передает его сеансовому уровню (уровень 5). Это продолжается до тех пор, пока сообщение со всеми его заголовками не достигнет физического уровня (уровень 1) и не будет передано получателю. На получающей стороне каждый уровень выполняет всю необходимую обработку и, удалив соответствующий заголовок, передает сообщение вверх на следующий уровень. В конце концов приложение-получатель получает доступ к исходным данным, отправленным приложением первого компьютера:



Теперь, когда нам стала понятна концепция семи уровней, мы можем рассмотреть функциональность каждого уровня более подробно. Начнем с самого низа и будем продвигаться вверх.

**Физический уровень** содержит физическую среду, в том числе требования к кабелям, соединители, спецификации интерфейсов, спецификации концентраторов, повторителей и т. д. На этом уровне точно определяется, какой физический сигнал будет использоваться для посылки "1", а какой будет представлять "0".

**МАС-адрес**, о котором уже рассказывалось в предыдущей статье, относится к уровню 2. Узлы локальной сети отправляют друг другу сообщения, используя IP-адреса, а они должны транслироваться в соответствующие МАС-адреса канальным уровнем. Протокол разрешения адресов (Address Resolution Protocol, ARP) транслирует IP-адреса в МАС-адреса. Кэш, содержащий известные МАС-адреса, ускоряет этот процесс, и его можно исследовать с помощью утилиты arp. Команда arp -а показывает МАС-адреса всех недавно использованных узлов в кэше ARP.

**Утилита arp** также позволяет установить соответствие между IP-адресами и МАC-адресами, с тем чтобы ARP-запросы для МАС-адресов больше не требовались. Однако это соответствие будет нарушено при замене сетевой платы, поэтому им следует пользоваться осторожно.

Другие обязанности канального уровня состоят в отправке и получении сообщений и обнаружении ошибок. В Ethernet также требуется обнаруживать конфликты, что мы уже обсуждали.

Коммутатор сети действует на канальном уровне, фильтруя сообщения в соответствии с МАС-адресами их получателей.

Уровнем выше над канальным уровнем находится сетевой уровень. На уровне 3 для соединения с другими узлами используется логическая адресация. МАС-адреса уровня 2 могут использоваться только внутри локальной сети, а обращаясь к узлам глобальной сети, надо использовать адресацию уровня 3. Internet Protocol (IP) — это протокол уровня 3; для идентификации узлов сети он использует IР-адреса. Маршрутизаторы выполняют маршрутизацию трафика между сетями на уровне 3.

На сетевом уровне хосты идентифицируются логическими адресами. На транспортном уровне идентифицируется приложение через так называемую конечную точку (endpoint). В протоколе TCP конечная точка задается комбинацией номера порта (port number) и IP-адреса.

**Транспортный уровень** различается в зависимости от того, используем ли мы надежную или ненадежную связь. При надежной связи, если сообщение было отправлено, но не было корректно получено, вырабатывается ошибка, в то время как при ненадежной связи сообщение отправляется, но никакой проверки, было ли оно вообще получено, не выполняется. При надежной связи транспортный уровень отвечает за отправку подтверждений на пакеты данных, что позволяет повторно передавать сообщения в случае искажения или потери данных, браковать дублирующие сообщения и т. д.

**Сетевая связь** на транспортном уровне может также различаться в зависимости от того, ориентирована ли она на соединения, или соединения отсутствуют:

* При связи, ориентированной на соединения, до отправки или получения сообщения требуется установить соединение.
* При связи без соединений устанавливать отдельные соединения необязательно и сообщения отправляются немедленно.

**Протокол TCP** использует ориентированный на соединения механизм связи, в то время как в UDP (User Datagram Protocol) используется механизм связи без организации соединений. Ориентированная на соединения связь является надежной, поскольку в этом случае отправляются подтверждения и сообщения посылаются повторно, если данные не получены или по какой-либо причине они были искажены.

Связь без установления соединений может быть полезна при широковещательной передаче, когда сообщения отправляются нескольким узлам. В этом случае доставка сообщения не гарантируется. Если необходим надежный обмен сообщениями, надежность может обеспечить протокол более высокого уровня, подключенный поверх механизма без установления соединения.

В модели OSI сеансовый уровень определяет обслуживание приложения, например, вход в приложение и выход из приложения. Сеанс представляет собой виртуальное (логическое) соединение между приложениями. Соединение сеансового уровня не зависит от расположенного ниже физического соединения на транспортном уровне, виртуальное соединение может существовать дольше, чем соединение на транспортном уровне. Для одного соединения сеансового уровня может потребоваться несколько соединений транспортного уровня.

Эти функциональные возможности мы сравниваем с возможностями, которые предоставляют сеансовые объекты ASP.NET. Сеансовые объекты существуют, пока сеанс не закончится по тайм-ауту (обычно через 20 минут), независимо от ТСР-соединений более низкого уровня.

Представительский уровень используется для форматирования данных в соответствии с требованиями приложений. На этом уровне обычно выполняются шифрование, дешифрование и сжатие.

Прикладной уровень—самый верхний уровень модели OSI. Этот уровень содержит приложения, в которых используются сетевые средства. Приложения могут выполнять такие задачи, как передача файлов, печать, электронная почта, Web-браузинг и т. д. Учебные приложения, которые мы будем создавать, принадлежат этому уровню.

**IP – Internet Protocol**

Набор протоколов TCP/IP разделен на уровни гораздо проще, чем предусмотрено моделью OSI. TCP и UDP - это транспортные протоколы, соответствующие уровню 4 OSI. Они используют IP, протокол уровня 3 OSI (сетевого уровня). Кроме этих трех протоколов, в наборе протоколов TCP/IP есть еще два базовых протокола, расширяющих IP: ICMP и IGMP. Функциональные возможности этих протоколов должны быть реализованы в уровне, содержащем IP.

**Internet Protocol** соединяет два узла. Каждый узел идентифицируется 32-битным адресом, называемым IP-адресом. При отправке сообщения IP-протокол получает его от протоколов верхнего уровня, TCP или UDP, и добавляет IP-заголовок, содержащий информацию о хосте-адресате.

Чтобы понять протокол IP, самый лучший способ — детально исследовать IP-заголовок. Содержащаяся в нем информация приведена в таблице:

|  |  |  |
| --- | --- | --- |
| Поле | Длина | Описание |
| Версия IP | 4 бита | Версия протокола IP, создавшего заголовок. Текущая версия протокола IP — 4. |
| Длина IP-заголовка | 4 бита | Длина заголовка. Минимальное значение — 5 в единицах по 32 бита, или 4 байта. Следовательно, минимальная длина заголовка равна 20 байтам. |
| Тип обслуживания | 1 байт | Поле типа обслуживания позволяет отправлять сообщения с нормальной или высокой производительностью, нормальной или увеличенной задержкой, нормальной или высокой надежностью. Это поле полезно при отправке в сеть дейтаграмм. Несколько разновидностей сетей используют эту информацию, чтобы выделить приоритет определенного трафика. Кроме того, сообщения управления сетью по сравнению с обычными сообщениями имеют повышенные приоритет и надежность. |
| Общая длина | 2 байта | В этих двух байтах задается общая длина сообщения — заголовка и данных— в октетах. Максимальный размер IP-пакета равен 65 535 байтов, но для большинства сетей такой размер непрактичен. Самый большой размер, который может быть принят всеми хостами, равен 576 байтам. Длинные сообщения могут разделяться на фрагменты — такой процесс называется фрагментацией. |
| Идентификация | 2 байта | Если сообщение разбито на фрагменты, поле идентификации помогает собрать фрагменты сообщения. Все фрагменты одного сообщения имеют один и тот же идентификационный номер. |
| Флаги | 3 бита | Эти флаги указывают, фрагментировано ли сообщение и является ли текущий пакет последним фрагментом сообщения. |
| Смещение фрагмента | 13 битов | В этих 13 битах задается смещение фрагментированного сообщения. Фрагменты могут поступать не в том порядке, в каком они были отправлены, поэтому смещение необходимо, чтобы восстановить исходные данные. Первый фрагмент сообщения имеет длину О, а в остальных фрагментах дается смещение, по которому следует поместить фрагмент. Единица смещения равна 8 байтам, так что значение смещения 64 означает, что второй фрагмент нужно присоединить к сообщению после 512 байтов первого пакета. |
| Время жизни | 1 байт | Значение "время жизни" (TTL) задает число секунд, которое сообщение может существовать, прежде чем будет отброшено. В этом значении необязательно указывается число секунд, поскольку каждый маршрутизатор, пересекаемый сообщением, должен уменьшить значение TTL на 1, даже если он затратил на обработку сообщения меньше одной секунды. Поэтому на практике в этом значении задается число допустимых "прыжков". |
| Протокол | 1 байт | В этом байте указывается протокол, используемый на следующем уровне стека протоколов для этого сообщения. Номера протоколов определены в доступной оперативной базе данных [Internet Assigned Number Authority (IANA)](http://www.iana.org/assignments/protocol-numbers). |
| Контрольная сумма заголовка | 2 байта | Это контрольная сумма одного заголовка. Поскольку заголовок изменяется с каждым отправленным сообщением, контрольная сумма также изменяется. |
| Адрес источника | 4 байта | В этом поле указывается 32-битный IP-адрес отправителя. |
| Адрес назначения | 4 байта | Это 32-битный IP-адрес, по которому отправлено сообщение. |
| Опции | переменная | Здесь могут появляться необязательные поля. Например, можно указать, что это сообщение секретно или совершенно секретно. Также предусмотрена возможность будущих расширений. |
| Дополнение | переменная | Это поле содержит переменное число нулей, такое, чтобы заголовок заканчивался на 32-битной границе. |

Internet Protocol (IP) определен в RFC 791. Документы [RFC (Request for Comments)](http://www.letf.org/rfc.html) содержат техническую информацию о многих важных интернет-технологиях.

**IP-адрес**

Каждый узел в сети TCP/IP может быть идентифицирован 32-битным IP-адресом. Обычно IP-адрес представляется четырьмя десятичными значениями в таком виде: 192.168.0.1. Каждое из этих чисел представляет собой один байт IP-адреса и может находиться в пределах от 0 до 255.

IP-адрес содержит две части: сетевую часть и часть хоста. В зависимости от класса сети сетевая часть состоит из одного, двух или трех байтов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Байт 1 | Байт 2 | Байт 3 | Байт 4 |
| A | Сеть (1—126) | Хост (0-255) | Хост (0—255) | Хост (0—255) |
| B | Сеть (128—191) | Сеть (0—255) | Хост (0—255) | Хост (0—255) |
| C | Сеть (192—223) | Сеть (0—255) | Сеть (0—255) | Хост (0—255) |

Первый бит адреса сети класса А должен быть 0, поэтому первый байт для сети класса А имеет двоичные значения в пределах от 00000001 (1) до 01111110 (126). Остальные три байта служат для идентификации узлов в сети, позволяя соединить в сети класса А более 16 млн. устройств.

Заметим, что в приведенной таблице адреса с числом 127 в первом байте пропущены, поскольку это зарезервированный диапазон адресов. Адрес 127.0.0.1 — это всегда адрес локального хоста, а 127.0.0.0 — адрес локальной обратной связи. Обратная связь используется для тестирования стека сетевых протоколов на одной машине, без прохода через сетевую интерфейсную плату.

В IP-адресе для сети класса В первые два бита всегда имеют значение 10, что дает диапазон от 10000000 (128) до 10111111 (191). Второй байт продолжает идентификацию сети значением от 0 до 255, оставляя два последних байта для идентификации узлов сети, всего до 65 534 устройств.

Сети класса С отличаются IP-адресом, в котором в первых трех битах установлено значение 110, разрешая значения в диапазоне от 11000000 (192) до 11011111 (223). В сети этого типа лишь один байт оставлен для идентификации узлов, поэтому к ней можно подсоединить только 254 устройства.

Число устройств, которое можно подсоединить к сети каждого из этих классов с особыми IP-адресами, обратно пропорционально числу возможных сетей этого типа. Например, сеть класса А, допуская 16 млн. хостов, оставляет только часть первого байта для идентификации сети. В результате во всем мире может существовать лишь 126 сетей класса А. Только крупные компании, подобные AT & Т, IBM, Xerox и HP, имеют такой сетевой адрес. Когда компания запрашивает IP-сеть в органе, ведающем сетями, обычно она получает сеть класса С.

Если компания пожелает, чтобы больше хостов напрямую были подключены к Интернету, можно найти еще одну сеть класса С. Если для каждого хоста в сети не требуется прямого доступа к Интернету, можно использовать частный IP-адрес, и тогда применяется другая опция.

Сетевые адреса классов А, В и С оставляют свободными адреса, имеющие в первом байте значения от 224 до 255.

Агентство IANA выделяет номера сетей и публикует их перечень на странице [http://www.iana.org/assignments/ipv4-adclress-space](http://www.iana.org/assignments/ipv4-address-space). Почти во всех странах есть региональные регистрационные ведомства, выдающие по запросам номера сетей. Региональные ведомства получают диапазон сетей от IANA.

Чтобы избежать исчерпания IP-адресов, хосты, не соединенные напрямую с Интернетом, могут использовать адреса из диапазонов частных адресов. Частные адреса уникальны не глобально, а только локально, внутри сети. Во всех классах сетей резервируются определенные диапазоны, которые могут использоваться как частные адреса хостами, не требующими непосредственного двустороннего доступа к Интернету. Такие хосты вполне могут обращаться к Интернету через шлюз, который не посылает во внешнюю сеть частный IP-адрес.

**Подсети**

Для соединения двух узлов в разных сетях требуется маршрутизатор. Номер хоста определяется 24 битами IP-адреса класса А, в то время как для сети класса С доступно лишь 8 битов. Маршрутизатор разделяет номер хоста на номер подсети и номер хоста в подсети. Включение дополнительных маршрутизаторов сократит объемы широковещательной передачи в сети, а это может сократить нагрузку в сети.

Новые маршрутизаторы главным образом включаются, чтобы улучшить возможность соединения между группами компьютеров в разных зданиях, городах и т. д. Рассмотрим пример разделения сети класса С с адресом 194.180.44 на подсети.

Такая сеть может фильтровать адреса с помощью маски подсети (subnet mask) 255.255.255.224. Первые три байта (состоящие из всех единиц) представляют собой маску для сети класса С. Последний байт - это десятичное значение двоичного представления 11100000, в котором первые три бита адреса хоста указывают подсеть, а последние пять битов представляют адрес хоста в конкретной подсети. Три бита подсети представляют 128, 64 и 32, и, таким образом, поддерживаются адреса подсетей, показанные ниже:



**IPv6**

Протокол, предшествовавший Internet Protocol, был разработан Управлением перспективных исследовательских работ Министерства обороны США (DARPA) в 1960-х годах, а набор протоколов TCP/IP получил признание лишь в 1980 г. Поскольку IP базировался на существовавших сетевых протоколах DARPA, он получил номер версии 4 и теперь известен как IPv4. В те времена, когда человечество в большинстве своем представляло себе мобильный телефон как трубку, которую можно снимать со стены и переносить к дивану, число хостов, поддерживаемых IP, казалось более чем достаточным.

Однако сегодня все хотят подключить к Интернету холодильники и газонокосилки, и IETF разработало новую версию IP - IPv6. Наиболее важное изменение этой версии по сравнению с IPv4 заключается в использовании для адресации не 32, а 128 битов, что позволит всем Tablet PC, Pocket PC, мобильным телефонам, телевизорам, автомобилям, газонокосилкам, кофеваркам и мусорным контейнерам стать полноправными хостами Интернета. :)

Кроме возможности назначить адрес почти каждому атому в Солнечной системе, в IPv6 появляется еще несколько полезных изменений:

* Возможности расширенной адресации. Чтобы определить диапазон адресов групповой рассылки, в адреса IPv6 может включаться маршрутная информация о группах. Кроме того, появляется альтернативный адрес для отправки сообщения любому хосту или любой группе хостов.
* Упрощение формата заголовка. Некоторые поля заголовка IPv4 удаляются, другие становятся необязательными. Однако полная длина заголовка IPv6 больше, чем в IPv4 из-за 128-битных адресов источника и назначения.
* Улучшенная поддержка расширяемости. В будущем добавлять расширения к протоколу IPv6 станет легче. Ограничения на длину для опций удалено.
* Маркирование потока. Для конкретных потоков трафика добавляется новая возможность. Поток - это последовательность пакетов, перемещающаяся от источника к назначению. В новом протоколе приложения могут предлагать аудио- и видеовозможности в реальном времени по различным потокам. Каждый поток может запрашивать обработку в реальном времени или особо качественную обработку у маршрутизаторов, через которые он распространяется.
* Аутентификация и секретность. Добавляются расширения IPv6, поддерживающие аутентификацию, секретность и конфиденциальность отправляемых данных.

**Номера портов**

Для идентификации узлов сети протокол IP использует IP-адреса, а транспортный уровень (уровень 4) использует конечные точки для идентификации приложения. Чтобы указать конечную точку приложения, протоколы TCP и UDP вместе с IP-адресом используют номер порта.

Сервер должен предоставить известную конечную точку, с которой мог бы соединиться клиент, хотя номер порта может создаваться для клиента динамически. Номера портов TCP и UDP имеют длину 16 битов, их можно подразделить на три категории:

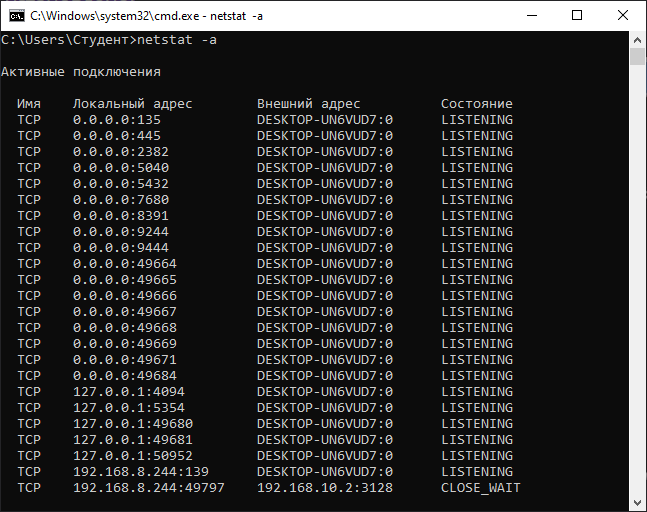
* Системные (известные) номера портов
* Пользовательские (зарегистрированные) номера портов
* Динамические, или частные, порты

**Системные номера портов** находятся в диапазоне от 0 до 1023. Эти номера должны использоваться только системными, привилегированными процессами. Широко известные протоколы пользуются номерами портов, установленными по умолчанию из этого диапазона.

**Пользовательские номера портов** находятся в диапазоне от 1024 до 49151. Ваше серверное приложение обычно будет занимать один из этих портов, и вы, если захотите сделать его известным сообществу пользователей Интернета, сможете зарегистрировать номер порта в IANA.

**Динамические номера портов** принимают значения из диапазона от 49 152 до 65 535. Если не требуется знать номер порта до запуска приложения, подойдет порт в этом диапазоне. Клиентские приложения, которые соединяются с серверами, могут использовать такой порт.

Запустив **утилиту netstat** с опцией -а, мы увидим перечень всех используемых в данный момент портов и указание о состоянии соединения — находится ли соединение в состоянии прослушивания или соединение уже было установлено:



В файле services из каталога <windir>\system32\drivers\etc перечислены многие предопределенные пользовательские и системные номера портов. Если порт содержится в перечне этого файла, то утилита netstat вместо номера порта отобразит имя протокола.

**Протоколы TCP и UDP**

**TCP— Transmission Control Protocol**

Обмен данными, ориентированный на соединения, может использовать надежную связь, для обеспечения которой протокол уровня 4 посылает подтверждения о получении данных и запрашивает повторную передачу, если данные не получены или искажены. Протокол TCP использует именно такую надежную связь. TCP используется в таких прикладных протоколах, как HTTP, FTP, SMTP и Telnet.

Протокол TCP требует, чтобы перед отправкой сообщения было открыто соединение. Серверное приложение должно выполнить так называемое пассивное открытие (passive open), чтобы создать соединение с известным номером порта, и, вместо того чтобы отправлять вызов в сеть, сервер переходит в ожидание поступления входящих запросов. Клиентское приложение должно выполнить активное открытие (active open), отправив серверному приложению синхронизирующий порядковый номер (SYN), идентифицирующий соединение. Клиентское приложение может использовать динамический номер порта в качестве локального порта.

Сервер должен отправить клиенту подтверждение (ACK) вместе с порядковым номером (SYN) сервера. В свою очередь клиент отвечает АСК, и соединение устанавливается.

После этого может начаться процесс отправки и получения сообщений. При получении сообщения в ответ всегда отправляется сообщение АСК. Если до получения АСК отправителем истекает тайм-аут, сообщение помещается в очередь на повторную передачу.

Поля заголовка TCP перечислены в следующей таблице:

Таблица: Заголовок TCP.

|  |  |  |
| --- | --- | --- |
| Поле | Длина | Описание |
| Порт источника | 2 байта | Номер порта источника |
| Порт назначения | 2 байта | Номер порта назначения |
| Последовательный номер | 4 байта | Последовательный номер генерируется источником и используется назначением, чтобы переупорядочить пакеты для создания исходного сообщения и отправить подтверждение источнику. |
| Номер подтверждения | 4 байта | Если установлен бит АСК поля "Управление", в данном поле содержится следующий ожидаемый последовательный номер. |
| Смещение данных | 4 бита | Информация о начале пакета данных. |
| Резерв | 6 битов | Резервируются для будущего использования. |
| Управление | 6 битов | Биты управления содержат флаги, указывающие, верны ли поля подтверждения (АСК), указателя срочности (URG), следует ли сбрасывать соединение (RST), послан ли синхронизирующий последовательный номер (SYN) и т. д. |
| Размер окна | 2 байта | В этом поле указывается размер приемного буфера. Используя подтверждающие сообщения, получатель может информировать отправителя о максимальном размере данных, которые тот может отправить. |
| Контрольная сумма | 2 байта | Контрольная сумма заголовка и данных; по ней определяется, был ли искажен пакет. |
| Указатель срочности | 2 байта | В этом поле целевое устройство получает информацию о срочности данных. |
| Опции | переменная | Необязательные значения, которые указываются при необходимости. |
| Дополнение | переменная | В поле дополнения добавляется столько нулей, чтобы заголовок заканчивался на 32-битной границе. |

TCP — это сложный, требующий больших затрат времени протокол, что объясняется его механизмом установления соединения, но он берет на себя заботу о гарантированной доставке пакетов, избавляя нас от необходимости включать эту функциональную возможность в прикладной протокол.

Протокол TCP имеет встроенную возможность надежной доставки. Если сообщение не отправлено корректно, мы получим сообщение об ошибке. Протокол TCP определен в RFC 793.

**UDP — User Datagram Protocol**

В отличие от TCP UDP — очень быстрый протокол, поскольку в нем определен самый минимальный механизм, необходимый для передачи данных. Конечно, он имеет некоторые недостатки. Сообщения поступают в любом порядке, и то, которое отправлено первым, может быть получено последним. Доставка сообщений UDP вовсе не гарантируется, сообщение может потеряться, и могут быть получены две копии одного и того же сообщения. Последний случай возникает, если для отправки сообщений в один адрес использовать два разных маршрута.

UDP не требует открывать соединение, и данные могут быть отправлены сразу же, как только они подготовлены. UDP не отправляет подтверждающие сообщения, поэтому данные могут быть получены или потеряны. Если при использовании UDP требуется надежная передача данных, ее следует реализовать в протоколе более высокого уровня.

Так в чем же преимущества UDP, зачем может понадобиться такой ненадежный протокол? Чтобы понять причину использования UDP, нужно различать однонаправленную передачу, широковещательную передачу и групповую рассылку.

Однонаправленное (unicast) сообщение отправляется из одного узла только в один другой узел. Это также называется связью "точка-точка". Протокол TCP поддерживает лишь однонаправленную связь. Если серверу нужно с помощью TCP взаимодействовать с несколькими клиентами, каждый клиент должен установить соединение, поскольку сообщения могут отправляться только одиночным узлам.

Широковещательная передача (broadcast) означает, что сообщение отправляется всем узлам сети. Групповая рассылка (multicast) - это промежуточный механизм: сообщения отправляются выбранным группам узлов.

UDP может использоваться для однонаправленной связи, если требуется быстрая передача, например для доставки мультимедийных данных, но главные преимущества UDP касаются широковещательной передачи и групповой рассылки.

Обычно, когда мы отправляем широковещательные или групповые сообщения, не нужно получать подтверждения из каждого узла, поскольку тогда сервер будет наводнен подтверждениями, а загрузка сети возрастет слишком сильно. Примером широковещательной передачи является служба времени. Сервер времени отправляет широковещательное сообщение, содержащее текущее время, и любой хост, если пожелает, может синхронизировать свое время с временем из широковещательного сообщения.

Заголовок UDP гораздо короче и проще заголовка TCP:

Заголовок UDP

|  |  |  |
| --- | --- | --- |
| Поле | Длина | Описание |
| Порт источника | 2 байта | Указание порта источника для UDP необязательно. Если это поле используется, получатель может отправить ответ этому порту. |
| Порт назначения | 2 байта | Номер порта назначения |
| Длина | 2 байта | Длина сообщения, включая заголовок и данные. |
| Контрольная сумма | 2 байта | Контрольная сумма заголовка и данных для проверки |

UDP — это быстрый протокол, не гарантирующий доставки. Если требуется поддержание порядка сообщений и надежная доставка, нужно использовать TCP. UDP главным образом предназначен для широковещательной и групповой передачи. Протокол UDP определен в RFC 786.

**Протоколы ICMP и IGMP**

**ICMP — Internet Control Message Protocol**

**ICMP** — это протокол управления сообщениями в Интернете, используется IP-устройствами, чтобы информировать другие IP-устройства о действиях и ошибках в сети. Без TCP IP не является надежным протоколом: он не отправляет подтверждения, не проверяет данные на ошибки (только контрольную сумму заголовка) и не повторяет передачу.

Об ошибках можно информировать с помощью сообщений ICMP. Сообщения ICMP используются для отправки ответной реакции о состоянии сети. Например, маршрутизатор, не найдя подходящего элемента для сети в таблице маршрутизации, отправляет сообщение ICMP "недостижимый пункт назначения". Найдя лучший путь, маршрутизатор может послать сообщение ICMP "перенаправить".

ICMP не располагается поверх IP, как могло бы показаться, напротив, сообщения ICMP отправляются внутри заголовка IP. Следовательно, протокол ICMP должен быть реализован модулем IP стека сети. Эти поля сообщения ICMP ставятся в начале заголовка IP:

|  |  |  |
| --- | --- | --- |
| Длина | Поле | Описание |
| 1 байт | Тип | В этом поле задается тип сообщения ICMP. Например, значение типа, равное 3, означает, что пункт назначения недостижим, 11 определяет, что время истекло, 12 — обнаружены некорректные параметры заголовка. |
| 1 байт | Код | Код предоставляет дополнительную информацию о типе сообщения. Для типа "недостижимый пункт назначения" код указывает, что именно недостижимо: сеть (0), хост (1), протокол (2) или порт (3). |
| 2 байта | Контрольная сумма | Контрольная сумма сообщения ICMP. |
| 4 байта | Зависит от типа | В последних 4 байтах заголовка ICMP может предоставляться дополнительная информация, зависящая от типа сообщения. |

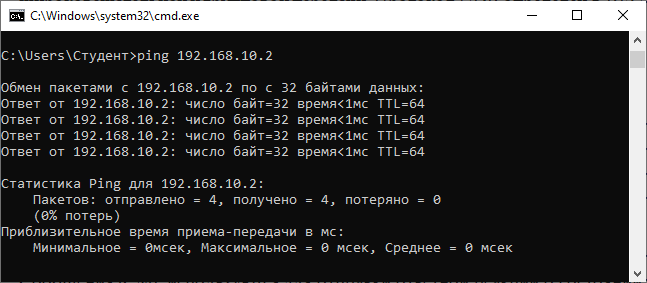
**Команда ping**

Утилита ping командной строки Windows отправляет ICMP-сообщение "эхо" целевому устройству, указанному именем хоста или IP-адресом в команде ping. Если устройство достижимо, обратно отправляется сообщение "отклик на эхо".

Эта команда очень полезна, когда нужно проверить, можно ли достичь данного устройства, есть ли проблемы на пути к нему (команда PING -t продолжает посылать сообщения, пока не будет остановлена) и как быстро сообщение доходит до устройства.

Если вы не можете достичь какого-либо хоста с использованием команды ping, то это не означает, что его нельзя достичь с помощью других протоколов. ICMP-сообщения "эхо"могут блокироваться маршрутизаторами или брандмауэрами.

На следующем снимке экрана показан вывод, создаваемый командой ping для хоста с IP-адресом 192.168.10.2. По умолчанию ping посылает узлу назначения четыре ICMP-сообщения и ждет откликов. На снимке показано, что были отправлены 32 байта данных и время, при этом не было получено отклика, поэтому все 4 пакета были потеряны:



**IGMP — Internet Group Management Protocol**

Как и ICMP, IGMP представляет расширение протокола IP и должен быть реализован модулем IP. IGMP используется приложениями групповой рассылки. При отправке широковещательного сообщения в локальной сети каждый узел этой сети анализирует сообщение вплоть до транспортного уровня, чтобы проверить, хочет ли какое-либо приложение получать сообщения от порта широковещательной передачи. Если ожидающие приложения отсутствуют, сообщение уничтожается и не передается выше транспортного уровня. Это означает, что каждому хосту нужно затратить несколько циклов центрального процессора (ЦП) независимо от того, интересует его широковещательное сообщение или нет.

При групповой рассылке этот момент учитывается, и сообщения отправляются не всем узлам локальной сети, а только группе узлов. Сетевая интерфейсная плата может определить, интересуется ли система данным сообщением, анализируя широковещательный МАС-адрес и не прибегая к помощи ЦП.

Заинтересованность в сообщениях групповой рассылки регистрируется отправкой запроса на членство в группе через IGMP-сообщение. Аналогично можно использовать IGMP, чтобы отказаться от членства.

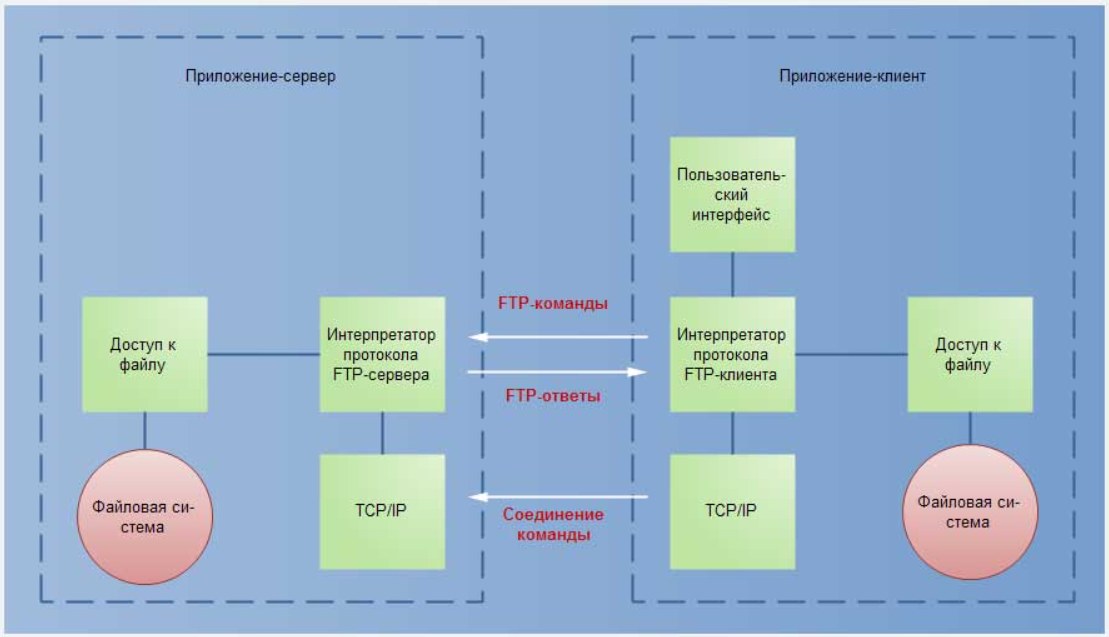
**Протоколы интернета и электронной почты.**

После обсуждения базовых протоколов мы можем подняться на более высокий уровень. Протоколы HTTP и FTP охватывают уровни 5—7 модели OSI.

**FTP — File Transfer Protocol**

FTP используется для копирования файлов с сервера и на сервер, а также для получения списка файлов и каталогов на сервере. FTP — это протокол прикладного уровня, базирующийся на TCP. Команды FTP включаются в блок данных TCP-сообщения.

Модель приложения с FTP-сервером и клиентом проиллюстрирована на следующем рисунке. Приложение-клиент представляет пользовательский интерфейс и создает FTP-запрос в соответствии с запросом пользователя и спецификацией FTP. FTP-команда посылается приложению-серверу через TCP/IP, и интерпретатор на сервере соответственно интерпретирует FTP-команду. В зависимости от FTP-команды в FTP-ответе клиенту возвращается с сервера список файлов или конкретный файл:



Протокол FTP имеет следующие характеристики:

* Надежная передача данных через TCP
* Анонимный доступ или аутентификация пользователя по имени и паролю
* Файлы отправляются в ASCII-коде в форме, поддерживаемой целевой платформой, или как неизмененные двоичные данные.

FTP-команды можно сгруппировать в следующие категории:

**Команды контроля доступа**

В FTP-командах контроля доступа указывается имя пользователя (USER) и пароль (PASS), установки могут изменяться (REIN), и соединение может быть закончено (QUIT).

**Команды параметров передачи**

FTP-передачу можно конфигурировать с помощью команд параметров передачи. Эти команды поддерживают изменение ASCII-кода на двоичный код, сжатие данных, изменение портов для отправки.

**Команды FTP-сервиса**

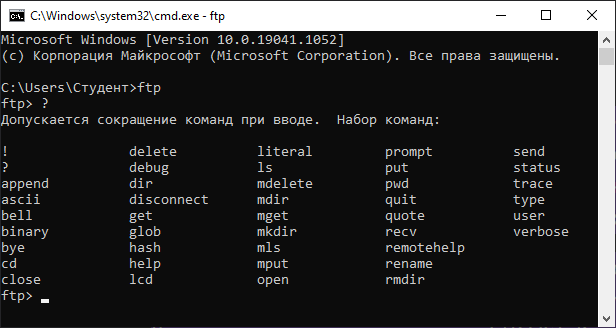
Копирование файлов с сервера (RETR), копирование файлов на сервер (STOR), удаление файлов (DELE), переименование файлов (RNTO), создание каталогов (MKD) и запрос списка файлов (LIST) — вот некоторые команды FTP-сервиса.

Протокол FTP определен в RFC 959.

**FTP-клиенты**

Чтобы понять суть протокола FTP, лучше всего поработать из командной строки с утилитой ftp, как показано на следующем рисунке. Программа ftp работает через приглашение ftp, позволяющее вводить команды. Эти команды отличаются от команд протокола FTP — вы можете увидеть их все, если введете команду ?. При введении команды open ftp.microsoft.com создается соединение с хостом ftp.microsoft.com.

Установка имени пользователя anonymous означает пользователя-гостя. Ответ 230 от сервера указывает, что соединение установлено и можно командой dir получить список файлов на сервере. Получив команду dir, программа ftp отправляет на сервер команду FTP LIST. Команда cd может применяться для смены директории на сервере, а команда get копирует файл клиенту, отправляя команду FTP RETR. Для закрытия соединения утилита ftp использует команду bye:

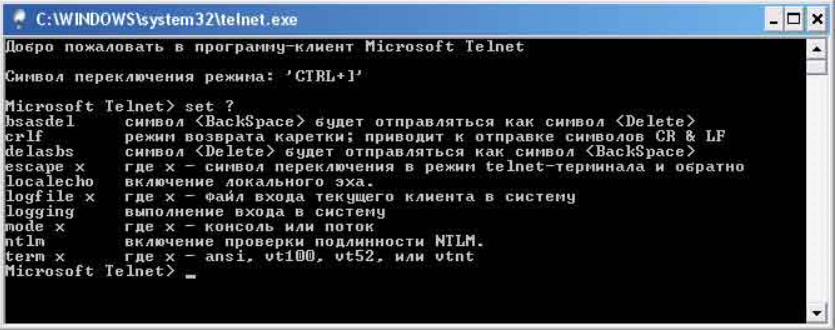


**HTTP — Hypertext Transfer Protocol**

HTTP — основной протокол, используемый Web-приложениями. Как и FTP, HTTP является надежным протоколом, и надежность его достигается благодаря использованию TCP. Как и FTP, HTTP также используется для передачи файлов через сеть. Но он в отличие от FTP обладает такими средствами, как кэширование, идентификация приложения-клиента, поддержка разных дополнений в формате MIME и т. д. Эти средства устанавливаются в заголовке HTTP.

Для демонстрации работы, выполняемой браузером Интернета, когда он запрашивает файлы из Web-сервера, мы можем имитировать браузер приложением telnet. Для запуска этого приложения введите telnet в диалоговом окне Run меню Start, и вы увидите приглашение Microsoft Telnet. Введите set local\_echo (set localecho для Windows XP), чтобы введенные программы отображались локально. Если не устанавливать эту опцию, команды, которые мы отправляем на сервер, не будут отображаться приложением telnet.

Теперь можно соединиться с Web-cepвeром командой open. Команда open msdn.microsoft.com 80 создает TCP-соединение с портом 80 сервера на msdn.microsoft.com. Приложение telnet использует по умолчанию порт 23, следовательно, надо указать порт для HTTP-запроса. По умолчанию HTTP-сервисы предоставляются портом 80 Web-сервера.



Как только соединение инициировано, можно послать на Web-cepвep HTTP-запрос. Простой запрос состоит из строки запроса, которую нужно завершить двумя нажатиями на клавишу Enter (две последовательности CR-LF). Такая строка запроса может выглядеть, как: GET /default.asp HTTP/1.0. Сервер возвращает HTTP-ответ, содержащий информацию о статусе.

Как видно, элементарный HTTP-запрос состоит лишь из одной строки. Однако полный HTTP-запрос будет состоять из строки запроса с дополнительными заголовками и данными.

В строке запроса можно указывать такие HTTP-команды, как GET, HEAD и POST. И GET, и POST запрашивают данные от сервера. Команда GET включает параметры запроса в URL, а в команде POST параметры находятся в блоке данных. Команда HEAD означает, что мы просто хотим узнать, когда был изменен запрашиваемый файл, чтобы проверить, находится ли в кэше новейшая его версия.

За строкой запроса могут следовать общие заголовки, заголовки запроса и заголовок сущности. Информация, помещаемая в заголовки, позволяет клиенту сообщать серверу об используемом браузере и предпочтительных языках, отправлять файл cookie или запрашивать только изменившиеся файлы. В примере с telnet мы уже видели некоторую информацию из заголовков, возвращенную сервером: дату, версию сервера, длину содержания, тип содержания и признаки управления кэшем.

**HTTPS — HTTP поверх SSL (Secure Socket Layer)**

Когда требуется обменяться с Web-сервером конфиденциальной информацией, можно воспользоваться протоколом HTTPS. HTTPS - это расширение протокола HTTP, и поэтому к нему применимы все принципы, которые обсуждались в предыдущем разделе. Однако в его основе лежит другой механизм, поскольку HTTPS использует SSL, первоначально разработанный компанией Netscape. SSL располагается поверх TCP и защищает сетевой обмен, используя принцип открытого/секретного ключа для обмена секретными симметричными ключами и симметричный ключ для шифрования сообщений.

Для поддержки HTTPS Web-сервер должен установить сертификат, чтобы его можно было идентифицировать. По умолчанию для HTTPS-запросов используется порт 443.

**SMTP — Simple Mail Transfer Protocol**

SMTP — это протокол, предназначенный для отправки и получения сообщений электронной почты. Он может использоваться для пересылки электронной почты между клиентом и сервером, использующими один и тот же транспортный протокол, или для пересылки сообщений между серверами, использующими разные транспортные протоколы. SMTP имеет возможность пересылать сообщения через среду транспортной службы. Однако SMTP не позволяет читать сообщения с почтового сервера, для этого используются протоколы POP3 и IMAP.

Служба SMTP составляет часть установки Internet Information Server в Windows 2000 и Windows XP.

Стандарт протокола SMTP определен в RFC 821; формат сообщений SMTP определен в RFC 822.

**POP3 — Post Office Protocol**

Протокол POP3 предназначался для отсоединенной среды. В небольших конфигурациях непрактично поддерживать постоянное соединение с почтовым сервером, например в такой среде, где время соединения нужно оплачивать. При использовании POP3 клиент может обращаться к серверу и извлекать сообщения, которые хранит для него сервер. Когда сообщения считываются клиентом, они обычно (но необязательно) удаляются с сервера.

Windows .NET Server включает сервер POP3. Протокол POP3 определен в RFC 1081.

**IMAP — Internet Message Access Protocol**

Как и РОРЗ, протокол IMAP предназначен для доступа к почте на почтовом сервере. Аналогично клиентам POP3 клиент IMAP может работать в автономном режиме, в котором почта обрабатывается на локальной машине. По сравнению с клиентами POP3 клиенты IMAP обладают более широкими возможностями в оперативном режиме, например, они могут извлекать только заголовки или только основные части указанных почтовых сообщений, искать конкретные сообщения на сервере и устанавливать флаги, например флаг "ответ отправлен". По существу, IMAP позволяет клиенту обрабатывать удаленный почтовый ящик, как если бы он был локальным.

Протокол IMAP определен в RFC 1730.

**NNTP — Network News Transfer Protocol**

NNTP - это протокол прикладного уровня для передачи, ретрансляции и извлечения сообщений, являющихся частью обсуждений в группах новостей. Этот протокол обеспечивает приложения-клиенты доступом к серверу новостей для извлечения выбранных сообщений и поддерживает передачу сообщений между серверами.

Протокол NNTP определен в RFC 850, 977 и 1036.

**Задание 1**. Проделать выше приведенные упражнеия.

**Часть 2. Работа с сетью**

**Пространство имен System.Net**

Пространство имен System.Net содержит сетевые классы для поиска IP-адресов, сетевой аутентификации, разрешений, отправки и получения данных. Рассмотрим эти классы, рассортировав их по группам.

**Поиск имен**

Чтобы получить IP-адрес из DNS-имени хоста или получить имя хоста из IP-адреса, можно использовать класс Dns. Класс DnsPermission представляет разрешение, необходимое для поиска имени. DnsPermissionAttribute — это класс атрибута, позволяющий отмечать сборки, классы и методы, нуждающиеся в этих полномочиях.

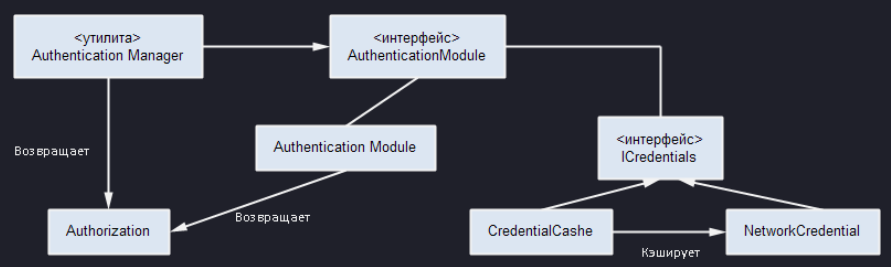
**IP-адреса**

IP-адреса обрабатываются в классе IPAddress. У одного хоста может быть несколько IP-адресов и алиасов. Вся эта информация содержится в классе IPHostEntry. Когда мы ищем имя, класс Dns возвращает объект типа IPHostEntry.

**Аутентификация и авторизация**

В классе AuthenticationManager есть статические методы, предназначенные для аутентификации клиентского пользователя. В этом классе-утилите используются модули, реализующие интерфейс IAuthenticationModule. Класс AuthenticationManager обращается к этим модулям, чтобы идентифицировать пользователя. Модули аутентификации получают информацию запроса и данные о личности пользователя с помощью интерфейса ICredentials и возвращают объект Authorization для авторизованных пользователей, которым разрешается использовать ресурс.

Приложение-клиент может передать данные о пользователе на сервер экземпляром класса NetworkCredential. Эти данные о пользователе могут быть занесены в кэш BCredentialCache:



**Запросы и ответы**

Абстрактные базовые классы, предназначенные для отправки запросов на сервер и получения ответов, называются WebRequest и WebResponse. В пространстве имен System.Net имеется несколько специальных реализаций этих классов для HTTP и доступа к файлам: HttpWebRequest, HttpWebResponse, FileWebRequest и FileWebResponse.

Классы HttpXXX также используют еще один класс из пространства имен System.Net: **класс HttpVersion** применяется для указания версии HTTP. У классов **HttpWebRequest** и **HttpWebResponse** есть свойство **ProtocolVersion**, определяющее версию HTTP - HttpVersion.Version10 или HttpVersion.Version11. Версия HTTP 1.0 использовалась на заре Интернета и продолжает использоваться некоторыми Web-серверами, текущая версия HTTP 1.1 включает некоторые дополнительные возможности, например, может поддерживать открытое соединение для нескольких запросов.

Разрешения, необходимые для классов запросов и ответов, определены в классе WebPermission и классе атрибута WebPermissionAttribute.

Класс компонентов WebClient облегчает использование WebRequest и WebResponse из Visual Studio .NET. Этот класс порожден от класса Component и поэтому может использоваться с функцией буксировки мышью из панели Toolbox. Однако по умолчанию он не сконфигурирован для использования Toolbox. С классом WebClient нетрудно копировать файлы с сервера и на сервер.

**Управление соединениями**

Классы **ServicePoint** и **ServicePointManager** играют важную роль для НТТР-соединений. Для ресурса экземпляр класса ServicePoint связан с URI и может обрабатывать несколько соединений. Класс-утилита ServicePointManager управляет объектами ServicePoint, создавая новые или отыскивая существующие объекты.

Пропускную способность приложения, которое одновременно запрашивает множество данных с сервера, можно повысить, увеличив число соединений на базе приложения. Чтобы ограничить выделение сетевых ресурсов, по умолчанию максимальное число соединений с одним и тем же сервером устанавливается равным двум.

**Записи cookies**

**Cookies** — это хранящиеся на стороне клиента наборы данных, которые используются сервером для запоминания некоторой информации между запросами. Когда для запросов данных с сервера используется такой Web-браузер, как Internet Explorer, он сам управляет приемом и сохранением записей cookies и посылает их обратно Web-cepвepy. При создании специального приложения, запрашивающего данные с Web-сервера, посылающего записи cookies, можно их считывать в объект класса CookieCollection, возвращаемый свойством Cookies объекта HttpWebResponse.

С помощью класса CookieContainer передаются записи cookie серверу. Сама запись cookie представлена в классе Cookie.

**Сокеты**

Вместо Web-классов используются классы сокетов, при этом мы, приобретая дополнительные возможности и гибкость, сталкиваемся с определенной сложностью. Большинство классов, которые используются в программировании сокетов, можно найти в пространстве имен **System.Net.Sockets**.

Программирование сокетов не только позволяет осуществлять связь, ориентированную на соединения, как в случае с HTTP, но также и реализовывать связь без установления соединений, которая используется при групповой рассылке или широковещательной передаче с UDP. Программирование сокетов — чрезвычайно гибкий механизм, позволяющий пользоваться самыми разными протоколами: GGP, ICMP, IGMP, IPX, SPX.

Обзор наиболее важных классов из пространства имен System.Net закончен, и теперь более детально рассмотрим некоторые из этих классов.

**Работа с URI**

Каждый день мы используем *универсальные идентификаторы ресурса* (***Uniform Resource Identifiers*, URI**), когда что-то ищем в WWW. **URI** нужны, чтобы идентифицировать и запросить новый вид ресурса. Используя URI, можно обращаться не только к Web-страницам, но и к FTP-серверу, Web-сервису и локальным файлам.

Вместо URI часто используется термин *унифицированный указатель ресурса* (***Uniform Resource Locator***, **URL**). **URI**-общий термин, используемый для ссылок на ресурсы. **URL** - это URI, связанный с такими популярными схемами URI, как http, ftp и mailto. В технической документации термин URL больше не употребляется.

Еще один термин может быть вам уже известен - *унифицированное имя ресурса* (*Uniform Resource Name*, **URN**). **URN** - это стандартизированный URI, используемый для указания ресурса независимо от его расположения в сети.

Проанализируем части URI, ссылающегося на страницу Web-сайта компании Global Knowledge:

*http://www.globalknowledge.net:80/training/generic.asp?pageid=1078&country=DACH*

* Первая часть URI называется **схемой (scheme)**. Схема определяет пространство имен URI и может сузить синтаксис следующего за схемой выражения. Многие схемы названы по соответствующим протоколам (как http, ftp), которые они используют, но это не является обязательным. В нашем примере *идентификатором схемы* является **http**. Ограничитель схемы (// в этом примере) отделяет схему от остальной части URL.
* После ограничителя схемы следует **имя сервера** или **IP-адрес** в десятичной записи с точками, например www.globalknowledge.net.
* За именем сервера или IP-адресом находится *номер порта*, определяющий соединение с конкретным приложением на сервере. Если номер порта не задан, используется номер порта, устанавливаемый для этого протокола по умолчанию (например, порт 80 для HTTP).
* Путь определяет страницу (и каталог) запрошенного ресурса. Он необязательно представляет физический файл на сервере, а может создаваться динамически. В данном случае путь имеет вид /training/generic.asp.
* От пути символом ? отделена последняя часть этого URI, называемая запросом (query). В нашем примере запрос определен строкой pageid=1078&country=DACH. Строка запроса может состоять из нескольких компонентов, каждый из которых задает переменную и значение, объединенные символом &. Несколько компонентов запроса могут комбинироваться символом &. Так, в нашем примере первый компонент — pageid=1078 с переменной pageid и значением 1078, а второй компонент - country=DACH.
* Разделы внутри ресурса можно отождествить с фрагментами. Фрагменты используются для ссылок на разделы внутри HTML-страницы. В разработке Web-страниц фрагменты также называются закладками (bookmarks). Символ # отделяет идентификатор фрагмента от пути. В URL http;//www.microsoft.com/net/basics/glossary.asp#NETFramework фрагментом является строка #NETFramework.

Если символ # добавлен в строку запроса, то это уже не фрагмент. В URL может присутствовать строка запроса или фрагмент, но не то и другое одновременно.

В URI зарезервировано использование нескольких символов — они не могут входить в имена хостов или путь, поскольку представляют собой специальные символы-разделители. В URI зарезервированы следующие символы:

***; / ? : @ & = + $ ,***

**Класс Uri** из пространства имен System инкапсулирует универсальный идентификатор ресурсов. Он содержит свойства и методы для анализа, сравнения и комбинирования URI.

Можно создать объект Uri, передав конструктору строку URI:

Uri baseURI = new Uri("http://professorweb.ru");

Если уже есть базовый объект Uri, можно создать новый URI, комбинируя базовый URI с относительным URI:

Uri baseURI = new Uri("http://professorweb.ru");

Uri newURI = new Uri(baseURI, "my/csharp/web/level2/2\_2.php");

Если базовый URI уже содержит путь, он игнорируется. В качестве базы для нового URI берутся лишь схема, порт и имя сервера.

В классе Uri есть несколько статических полей только для чтения, позволяющих получить некоторые широко распространенные схемы:

**Uri.UriSchemeFile**

Файловая схема используется для доступа к файлам локально или на совместно используемых сетевых ресурсах, для которых могут применяться имена, соответствующие соглашению об универсальном назначении имен (Universal Naming Convention, UNC).

**Uri.UriSChemeFtp**

Протокол FTP со схемой ftp используется для получения файлов с ftp-сервера и, напротив, помещения файлов на ftp-сервер.

**Uri.UriSchemeGopher**

Протокол gopher был предшественником HTTP. Он предоставлял возможности иерархического просмотра текстовой информации о содержании, в чем превосходил FTP. Но скоро был заменен протоколом HTTP.

**Uri.UriSchemeHttp, Uri.UriSchemeHttps**

Эти две схемы хорошо известны: http и https. Схема https используется для защищенного обмена.

**Uri.UriSchemeMailto**

Схема mailto используется для отправки почтовых сообщений.

**Uri.UriSchemeNews, Uri.UriSchemeNntp**

Схемы news и nntp применяются в группах новостей, использующих протокол NNTP.

В классе Uri есть статические методы для проверки правильности схемы и имени хоста: **Uri.CheckSchemeName()** возвращает true, если имя схемы задано правильно, а метод **UriCheckHostName()** не только проверяет имя хоста, но и возвращает значение перечисления ***UriHostNameType***, указывающее тип хоста.

В классе Uri имеется масса свойств с доступом только на чтение, позволяющих обращаться ко всем частям URI. В следующей таблице используем приведенный выше URI как пример, демонстрирующий применение свойств:

|  |  |
| --- | --- |
| AbsoluteUri | Это свойство показывает полный URI. Если указанный номер порта для протокола равен номеру порта по умолчанию, конструктор Uri автоматически его удаляет. Для нашего примера значение свойства AbsoluteUri выглядит так: *http://www.globalknowledge.net/t raining/generic.asp?pageid=1078&country=DACH*. Если конструктору класса Uri передается имя файла, свойство AbsoluteUri автоматически помещает перед именем файла схему file://. |
| Scheme | Схема — первая часть URI, и в данном случае это свойство возвращает значение http. |
| Host | Свойство Host показывает имя хоста из URI: www.globalknowledge.net |
| Authority | Если номер порта равен номеру, используемому протоколом по умолчанию, свойство Authority показывает ту же строку, что и свойство Host. Если используется другой номер порта, то свойство Authority также показывает номер порта. |
| HostNameType | Тип имени хоста зависит от используемого имени. В данном случае получается то же самое значение перечисления UriHostNameType, которое было рассмотрено выше. |
| Port | С помощью свойства Port получается номер порта — 80. |
| AbsolutePath | Абсолютный путь начинается после номера порта в URI и заканчивается перед строкой запроса. В этом случае он имеет значение /training/generic.asp. |
| LocalPath | Локальный путь дает значение /training/generic.asp. Как можно видеть, для запроса HTTP между AbsolutePath и LocalPath нет никакого различия. Различие появляется, если URI ссылается на совместно используемый сетевой ресурс. Для URI в виде file:\\server\share\directory\file.txt свойство LocalPath возвращает только имена directory и file, а свойство AbsolutePath включает имена server и share. |
| Query | Свойство Query показывает строку, следующую после пути: ?pageid=1078&country=DACH. |
| PathAndQuery | Свойство PathAndQuery дает комбинацию пути и строки запроса: /training/generic.asp?pageid=1078&country=DACH. |
| Fragment | Если после пути следует фрагмент, он возвращается в свойстве Fragment. За путем могут следовать только строка запроса или фрагмент. Фрагмент идентифицируется символом # |
| Segments | Свойство Segments возвращает массив строк, сформированный из пути. В данном случае у нас есть три сегмента: /, training/ и generic.asp. |
| UserInfo | Установленное в URI имя пользователя можно прочитать из свойства UserInfo. Передача имен пользователей распространена в протоколе FTP, и если указан не анонимный пользователь, например ftp://myuser@ftp.myserver.com, то свойство UserInfo вернет myuser. |

Кроме перечисленных, существует еще несколько свойств, возвращающих булевы значения, если URI представляет файл, путь UNC, адрес обратной связи или если для данного протокола используется номер порта по умолчанию. Это свойства IsFile, IsUnc, IsLoopback и IsDefaultPort соответственно.

После создания конструктором экземпляр класса Uri не может больше изменяться. Свойства класса Uri доступны только на чтение. Для динамического изменения URI можно использовать класс UriBuilder. Его свойства аналогичны свойствам класса Uri, но их значения можно как считывать, так и записывать. Благодаря доступу только на чтение класс Uri гораздо быстрее класса UriBuilder. С такой идеей вы, возможно, знакомы, если работали с классами String и [StringBuilder](https://professorweb.ru/my/csharp/charp_theory/level4/4_8.php).

В следующем примере передачей конструктору схемы, имени хоста, имени порта и пути создается экземпляр класса UriBuilder. Затем изменим путь, устанавливая свойство Path. Свойство Uri класса UriBuilder возвращает доступный только на чтение экземпляр класса Uri:

**Класс WebClient**

Если необходимо только запросить файл с определенного URI (Uniform Resource Identifier — унифицированный идентификатор ресурса), то простейшим в использовании классом .NET, который подходит для этого, будет System.Net.WebClient. Этот исключительно высокоуровневый класс предназначен для выполнения базовых операций с помощью всего одной или двух команд. В настоящее время в .NET Framework поддерживаются URI, начинающиеся с идентификаторов http:, https: и file:.

Важно отметить, что термин URL (Uniform Resource Locator — универсальный локатор ресурсов) больше не используется в новых технических спецификациях, а вместо него отдается предпочтение URI. URI имеет приблизительно тот же смысл, что и URL, но немного более общий, потому что в URL не подразумевается обязательное применение одного из знакомых протоколов, таких как HTTP или FTP.

**Загрузка файлов**

Для загрузки файлов с использованием WebClient доступны два метода. Выбор метода зависит от того, как должно обрабатываться содержимое файла. Если необходимо просто сохранить файл на диске, следует применять метод DownloadFile(). Этот метод принимает два параметра: URI файла и местоположение (путь и имя файла) для сохранения запрошенных данных:

WebClient client = new WebClient();

client.DownloadFile("https://narfu.ru/university/library/books/0690.pdf","Грошев А.С. Информатика.pdf");

Console.WriteLine("Файл загружен");

Часто приложение должно обрабатывать данные, извлеченные с веб-сайта. Это обеспечивает метод OpenRead(), возвращающий ссылку на Stream, которую можно использовать для извлечения данных в память:

WebClient client = new WebClient();

Stream stream = client.OpenRead("http://www.birskdo.ru");

В следующем примере демонстрируется применение метода WebClient.OpenRead(). Содержимое загруженной страницы будет отображено в консоле В начало файла к списку директив using потребуется добавить ссылки на пространства имен System.Net и System.IO.

static void Main(string[] args)

{

WebClient client = new WebClient();

Stream stream = client.OpenRead("http://www.birskdo.ru");

StreamReader sr = new StreamReader(stream);

string newLine;

while ((newLine = sr.ReadLine())!=null)

Console.WriteLine(newLine);

Console.WriteLine("Файл загружен!!!");

Console.ReadLine();

}

В этом примере класс StreamReader из пространства имен System.IO подключается к сетевому потоку. Это позволяет получить данные из потока в виде текста, используя высокоуровневые методы вроде ReadLine(). На рис. показаны результаты запуска этого кода:



**Задание 2.**

Создайте новый проект как стандартное приложение WPF и добавьте элемент управления TextBox. В начало файла к списку директив using потребуется добавить ссылки на пространства имен System.Net и System.IO. Затем добавьте обработчик клика по кнопке:

private void button1\_Click(object sender, RoutedEventArgs e)

{

WebClient client = new WebClient();

Stream stream = client.OpenRead("http://www.birskdo.ru");

StreamReader sr = new StreamReader(stream);

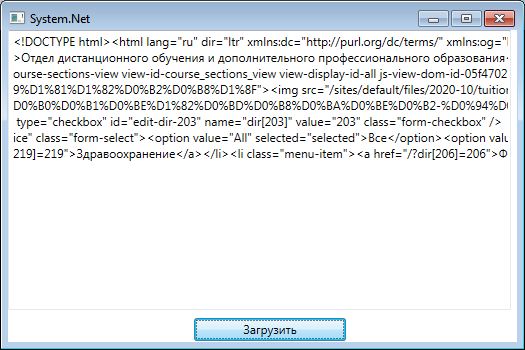
string newLine;

while ((newLine = sr.ReadLine()) != null)

textBox1.Text += newLine;

}

В этом примере класс StreamReader из пространства имен System.IO подключается к сетевому потоку. Это позволяет получить данные из потока в виде текста, используя высокоуровневые методы вроде ReadLine(). На рис. показаны результаты запуска этого кода:



Класс WebClient также включает в себя метод OpenWrite(). Этот метод возвращает записываемый поток для отправки данных по определенному URI. При этом можно указать метод, который должен использоваться для отправки данных на хост, по умолчанию это POST.

**Выгрузка файлов**

В классе WebClient также имеются методы UploadFile() и UploadData(). Они используются, когда нужно отправить HTML-форму или выгрузить на сервер целый файл. UploadFile() выгружает файл в указанное местоположение по указанному имени локального файла, в то время как UploadData() выгружает двоичные данные, представленные в виде массива байт, по указанному URI (есть также метод DownloadData(), предназначенный для извлечения массива байтов из URI).

WebClient client = new WebClient();

client.UploadFile("http://www.birskdo.ru","Ex.txt");

byte[] image;

client.UploadData("http://www.birsk.ru/", image);

Хотя класс WebClient очень прост в использовании, он обладает весьма ограниченными возможностями. В частности, невозможно передавать регистрационные данные для аутентификации — одна из проблем, связанных с загрузкой данных, состоит в том, что лишь относительно немного сайтов принимают файлы без аутентификации.

В запросы можно добавлять информацию заголовка и проверять заголовки в ответе, но лишь в очень общем смысле — специфическая поддержка какого-то одного протокола отсутствует. Причина в том, что WebClient — класс общего назначения, предназначенный для работы с любым протоколом, позволяющем отправлять запросы и получать ответы (вроде HTTP и FTP). Он не может обработать никаких средств, специфичных для какого-то одного протокола, например, сокеты, которые специфичны для HTTP. Чтобы воспользоваться преимуществами этих средств, нужно работать с семейством классов, основанным на двух других классах из пространства имен System.Net, а именно — WebRequest и WebResponse.

**Классы WebRequest и WebResponse**

Класс WebRequest представляет запрос информации для отправки по определенному URI. Идентификатор URI передается в качестве параметра методу Create(). Класс WebResponse представляет данные, извлекаемые из сервера. Вызов метода WebRequest.GetResponse() на самом деле приводит к отправке запроса веб-серверу и к созданию объекта WebResponse для просмотра возвращенных данных. Как и в случае объекта WebClient, можно получить поток для представления данных, но для этого должен использоваться метод WebResponse.GetResponseStream().

Сначала рассмотрим класс WebRequest:

Таблица Методы и свойства класса WebRequest.

|  |  |
| --- | --- |
| Методы и свойства | Описание |
| Create() и CreateDefault() | В классе WebRequest нет открытого конструктора. Вместо конструктора для создания экземпляров класса могут использоваться статические методы Create() и CreateDefault(). Эти методы в действительности создают не объект типа WebRequest, а новый объект класса, производного от WebRequest, такого как HttpWebRequest или FileWebRequest. |
| RegisterPrefix() | Используя метод RegisterPrefix(), можно зарегистрировать класс для обработки специфического протокола. Объекты этого класса будут создаваться методом WebRequest.Create(). Этот механизм называется "подключаемыми протоколами" (pluggable protocols). |
| GetRequestStream() | Метод GetRequestStream() возвращает объект потока, который может использоваться для отправки некоторых данных на сервер. |
| BeginGetRequestStream() и EndGetRequestStream() | Асинхронный доступ к потоку запроса выполняется методами BeginGetRequestStream() и EndGetRequestStream(). |
| GetResponse() | Метод GetResponse() возвращает объект WebResponse, который может использоваться для чтения данных, полученных от сервера. |
| BeginGetResponse() и EndGetResponse() | Как и для потока запроса, имеются асинхронные методы дпя получения потока ответа. |
| Abort() | Если метод BeginXX() начал асинхронную обработку, ее можно остановить методом Abort(). |
| RequestUri | RequestUri - свойство только для чтения, возвращающее URI, связанный с WebRequest. Этот URI может быть установлен в статическом методе Create() данного класса. |
| Method | Свойство Method используется, чтобы получить или установить метод для конкретного запроса. Объект HttpWebRequest поддерживает HTTP-методы GET, POST, HEAD и т. д. |
| Headers | В зависимости от используемого протокола серверу может передаваться и от сервера может получаться различная информация в заголовках. Информация заголовка протокола содержится в коллекции WebHeaderCollection, к которой можно обращаться через свойство Headers. |
| ContentType и ContentLength | Тип данных, отправленных серверу, определяется в свойстве ContentType. Могут быть разные типы данных такой длины, чтобы данные могли разместиться в массиве байтов. Тип содержания обычно определяет МIМЕ-тип данных: image/jpeg, image/gif, text/html или text/xml. |
| Credentials | Если серверу требуется аутентификация пользователя, удостоверения личности пользователя можно установить через свойство Credentials. |
| PreAuthenticate | Для протоколов, поддерживающих предварительную аутентификацию, в свойстве PreAuthenticate можно установить значение true. По умолчанию Web-браузер сначала пытается обратиться к странице Web-сайта без аутентификации. Если Web-сайту требуется аутентификация, сервер отвечает, что для неидентифицированных пользователей доступ отклонен. Следующий запрос, выполняемый клиентом, содержит информацию аутентификации. Этого дополнительного цикла обмена можно избежать, если установить в свойстве PreAuthenticate значение true. |
| Proxy | В свойстве Proxy можно установить Web-прокси, который используется для этого запроса. |
| ConnectionGroupName | В свойстве ConnectionGroupName можно определить пул соединений, который должен использоваться с этим объектом WebRequest. |
| Timeout | Свойство Timeout определяет время в миллисекундах, которое необходимо для ответа от сервера. По умолчанию установлено значение 100 000 млс. Если в течение этого времени сервер не отвечает, порождается исключение WebException. |

Класс WebResponse используется для чтения данных от сервера. Объект этого класса возвращается методом GetResponse(), как видно при рассмотрении класса WebRequest.

Таблица: Методы и свойства класса WebRequest

|  |  |
| --- | --- |
| Методы и свойства | Описание |
| GetResponseStream() | Метод GetResponseStream() возвращает объект потока, который используется для чтения ответа от сервера. |
| Close() | Если объект ответа больше не нужен, его следует закрыть методом Close(). |
| ResponseUri | С помощью свойства ResponseUri мы можем считать URI, связанный с объектом ответа. Он может совпадать с URI объекта WebRequest, но может и отличаться, если сервер переадресовал запрос к другому ресурсу. |
| Headers | Свойство Headers возвращает коллекцию WebHeaderCollection, которая включает специфичную для протокола информацию о заголовках, возвращаемых от сервера. |

**Задание 3. Выполнить упражнение, приведенного ниже примера**:

Рассмотрим небольшой пример использования этих классов в WPF-приложении:

<Grid**>**

<Grid**.**RowDefinitions**>**

<RowDefinition Height**="auto"/>**

<RowDefinition Height**="auto"/>**

<RowDefinition/**>**

<RowDefinition Height**="auto"/>**

<RowDefinition/**>**

</Grid**.**RowDefinitions**>**

<StackPanel Orientation**="Horizontal" Margin="8">**

<TextBlock VerticalAlignment**="Center">**URL-адрес</TextBlock**>**

<TextBox x**:**Name**="txb\_url" Margin="14,**0" Width**="250" VerticalContentAlignment="Center" Text="http://www.birsk.ru/"**/**>**

<Button Click**="request\_Click" Padding="5" Content="Получить информацию"/>**

</StackPanel**>**

<TextBlock Margin**="8" Text="Исходный код страницы:** " Grid**.**Row**="1"/>**

<TextBox x**:**Name**="txb\_sourceCode" Grid.**Row**="2" Padding="5" Margin="8" TextWrapping="Wrap" VerticalScrollBarVisibility="Visible"/>**

<GridSplitter Grid**.**Row**="3" HorizontalAlignment="Stretch" Height="3" Margin="8,**0" Background**="#aaa**"/**>**

<TextBox x**:**Name**="txb\_serverInfo" Grid.**Row**="4" Padding="5" Margin="8" TextWrapping="Wrap" VerticalScrollBarVisibility="Visible"/>**

</Grid**>**

**public** void request\_Click**(object** sender**,** RoutedEventArgs e**)**

**{**

// Создать объект запроса

WebRequest request **=** WebRequest**.**Create**(**txb\_url**.**Text**);**

// Получить ответ с сервера

WebResponse response **=** request**.**GetResponse**();**

// Получаем поток данных из ответа

**using** **(**StreamReader stream **=** **new** StreamReader**(**response**.**GetResponseStream**()))**

**{**

// Выводим исходный код страницы

string line**;**

**while** **((**line **=** stream**.**ReadLine**())** **!=** **null)**

txb\_sourceCode**.**Text **+=** line **+** "\n"**;**

**}**

// Получаем некоторые данные о сервере

string messageServer **=** "Целевой URL: \t" **+** request**.**RequestUri **+** "\nМетод запроса: \t" **+** request**.**Method **+**"\nТип полученных данных: \t" **+** response**.**ContentType **+** "\nДлина ответа: \t" **+** response**.**ContentLength **+** "\nЗаголовки"**;**

// Получаем заголовки, используем LINQ

WebHeaderCollection whc **=** response**.**Headers**;**

var headers **=** Enumerable**.**Range**(**0**,** whc**.**Count**)**

**.**Select**(**p **=>**

**{**

**return** **new**

**{**

Key **=** whc**.**GetKey**(**p**),**

Names **=** whc**.**GetValues**(**p**)**

**};**

**});**

**foreach** **(**var item **in** headers**)**

**{**

messageServer **+=** "\n " **+** item**.**Key **+** ":"**;**

**foreach** **(**var n **in** item**.**Names**)**

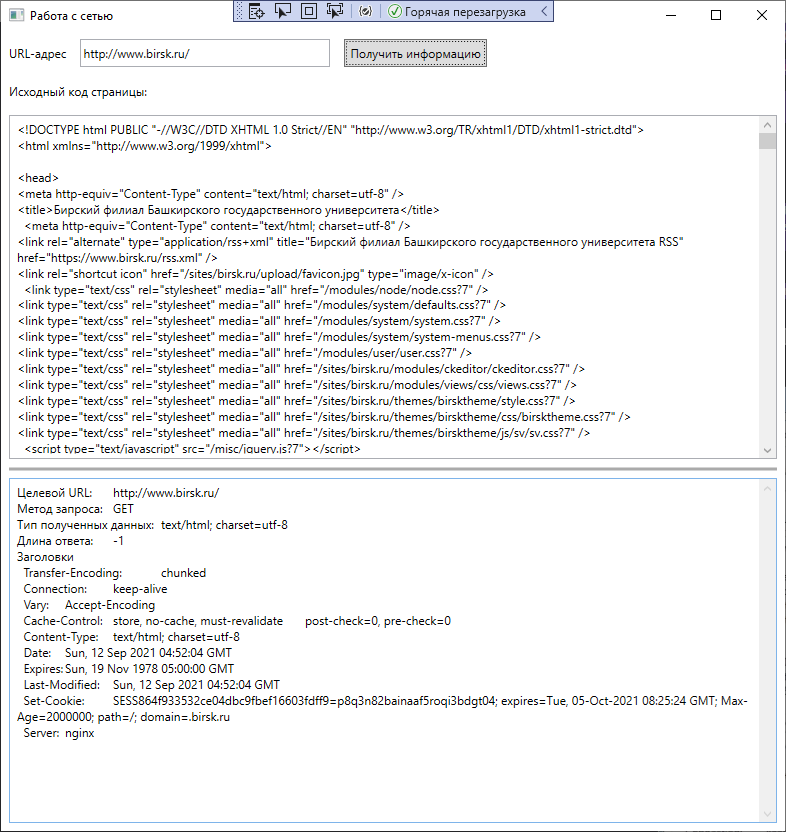
messageServer **+=** "\t" **+** n**;**

**}**

txb\_serverInfo**.**Text **=** messageServer**;**

**}**

Результат работы программы:



**Подключаемые протоколы**

WebRequest — это абстрактный класс, поэтому метод WebRequest.Create() не может создать объект типа WebRequest — вместо этого создается объект класса, производного от WebRequest. При передаче HTTP-запроса методу WebRequest.Create() создается объект HttpWebRequest. При передаче запроса со схемой файла создается объект FileWebRequest.

Как показано далее, схемы http, https и file предопределены в конфигурационном файле .NET, файле machine.config. Конфигурационный файл можно найти в каталоге <windows>\Microsoft.NET\Framework\<version>\CONFIG.

Набор протоколов, используемых классом WebRequest, можно расширить программно или добавив элемент в конфигурационный файл. Для поддержки нового протокола, отличного от схем http, https и file, нужно создать новый класс, производный от WebRequest, например FtpWebRequest для протокола FTP. Этот класс должен переопределить методы и свойства базового класса и в них реализовать специфичное для протокола поведение. Кроме того, требуется определить класс-инициатор (factory class), создающий объекты класса FtpWebRequest. Такой класс-инициатор, используемый классом WebRequest, должен реализовать интерфейс IWebRequestCreate. Назовем этот класс FtpWebRequestCreator. Экземпляр этого класса должен быть зарегистрирован для схемы ftp с помощью класса WebRequest:

WebRequest.RegisterPrefix("ftp", new FtpWebRequestCreator);

Если теперь схема ftp используется с методом WebRequest.Create(), создается и возвращается в программу новый экземпляр класса FtpWebRequest. Теперь объект request можно использовать для копирования файлов с FTP-cepвера и на FTP-сервер. Здесь мы не собираемся заниматься реализацией класса FtpWebRequestCreator, но вы можете это сделать самостоятельно. Для программирования FTP-клиента требуется использовать классы сокетов с соединением TCP.

**FileWebRequest и FileWebResponse**

Чтение и запись локальных файлов или файлов, находящихся на совместно используемых устройствах, не очень отличаются от чтения и записи файлов, расположенных на Web-серверах. Чтобы считывать и записывать файлы, используем классы FileWebRequest и FileWebResponse. Однако многие методы и свойства, определенные в базовых классах WebRequest и WebResponse, не используются в производных классах, и в документации MSDN они лишь перечисляются, как "зарезервированные для использования в будущем".

**Задание 4.**

Для демонстрации возможного использования классов FileWebRequest и FileWebResponse создается простое приложение WPF, в котором имя открываемого файла можно ввести в текстовом поле, после чего файл открывается и отображается в многострочном текстовом поле. В открытый файл можно будет записать новый текст:

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="auto"/>

<RowDefinition Height="auto"/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="auto"/>

<ColumnDefinition/>

<ColumnDefinition Width="auto"/>

</Grid.ColumnDefinitions>

<TextBlock Text="Открыть файл" VerticalAlignment="Center" Margin="8"/>

<TextBox x:Name="txb\_fileuri" Margin="14,8" VerticalContentAlignment="Center" Grid.Column="1"/>

<Button Content="Открыть" Click="openFile\_Click" Padding="5" MinWidth="120" Grid.Column="2" Margin="0,8,8,8"/>

<TextBlock Text="Записать в файл" VerticalAlignment="Center" Grid.Row="1" Margin="8"/>

<TextBox x:Name="txb\_writefile" Margin="14,8" VerticalContentAlignment="Center" Grid.Row="1" Grid.Column="1"/>

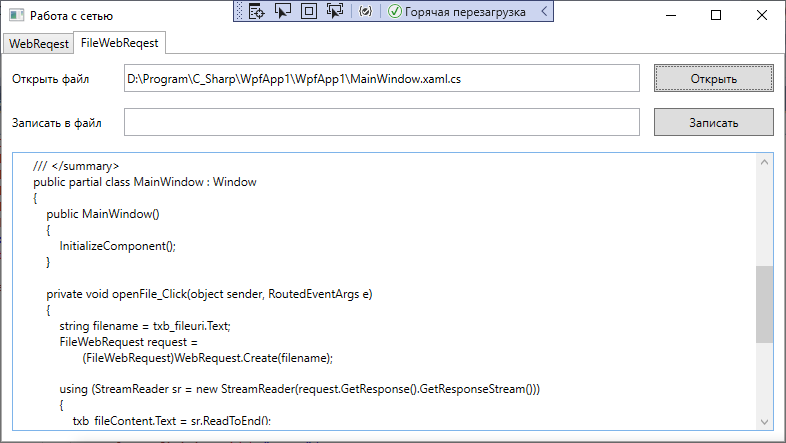
<Button Content="Записать" Click="writeFile\_Click" Padding="5" MinWidth="120" Grid.Row="1" Grid.Column="2" Margin="0,8,8,8"/>

<TextBox x:Name="txb\_fileContent" TextWrapping="Wrap" Margin="8" Padding="5" Grid.ColumnSpan="3" Grid.Row="2"

VerticalScrollBarVisibility="Visible"/>

</Grid>

Результат работы:



**Чтение из файлов**

Обработчик щелчка по кнопке "Открыть" открывает файл и записывает содержание файла в многострочное текстовое поле. Передадим имя файла методу WebRequest.Create(). Ставить схему file:// перед именем файла необязательно. Класс WebRequest создает объект Uri и использует его свойство AbsolutePath. Как указывалось ранее, класс Uri автоматически предпосылает имени файла корректную схему. Поэтому передача имени файла классу WebRequest создаст объект FileWebRequest , и требуется лишь привести его тип. Метод GetResponse() возвращает объект FileWebResponse, который сразу же используется для создания методом GetResponseStream() объекта Stream. Обычными методами класса StreamReader считывается поток. Данные всего файла считываем в строку и передаем ее в свойство Text многострочного текстового поля txb\_fileContent:

private void openFile\_Click(object sender, RoutedEventArgs e)

{

string filename = txb\_fileuri.Text;

FileWebRequest request =(FileWebRequest)WebRequest.Create(filename);

using (StreamReader sr = new StreamReader(request.GetResponse().GetResponseStream()))

{

txb\_fileContent.Text = sr.ReadToEnd();

}

}

**Запись в файлы**

Для записи данных обратно в файл реализуем обработчик щелчка по кнопке "Записать". Как и раньше, создадим объект WebRequest, передавая имя файла. Теперь вместо StreamReader используем StreamWriter. Кроме этого есть еще одно существенное изменение в коде. Чтобы сделать поток "записываемым", следует установить в свойстве Method значение "PUT". По умолчанию это свойство имеет значение "GET", указывая, что поток можно только считывать:

private void writeFile\_Click(object sender, RoutedEventArgs e)

{

WebRequest request = WebRequest.Create(txb\_fileuri.Text);

request.Method = "PUT";

using (StreamWriter sw = new StreamWriter(request.GetRequestStream()))

{

sw.Write(txb\_writefile.Text);

}

}

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. **Фленов М. Е.**Библия C#. — 4-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2019. — 512 с.: ил.
2. URL: https://www.interestprograms.ru/sources/csharp/os-windows/desktop/setevye-prilozheniya (дата обращения 10.04.2021)
3. URL: <https://professorweb.ru/my/csharp/web/level1/1_1.php>(дата обращения 10.04.2021)
4. URL: <https://professorweb.ru/my/csharp/web/level1/1_2.php> (дата обращения 10.04.2021)
5. URL: <https://professorweb.ru/my/csharp/web/level1/1_3.php> (дата обращения 10.04.2021)
6. URL: <https://professorweb.ru/my/csharp/web/level1/1_4.php> (дата обращения 10.04.2021)
7. URL: <https://professorweb.ru/my/csharp/web/level1/1_5.php> (дата обращения 10.04.2021)
8. URL: <https://professorweb.ru/my/csharp/web/level1/1_6.php> (дата обращения 10.04.2021)
9. URL: <https://professorweb.ru/my/csharp/web/level2/2_1.php> (дата обращения 10.04.2021)
10. URL: <https://professorweb.ru/my/csharp/web/level2/2_2.php> (дата обращения 10.04.2021)
11. URL: <https://professorweb.ru/my/csharp/web/level2/2_3.php> (дата обращения 10.04.2021)
12. URL: <https://professorweb.ru/my/csharp/web/level2/2_4.php> (дата обращения 10.04.2021)