2 ЛАБОРАТОРНА РОБОТА №2

КЛІЄНТ-СЕРВЕРНА ВЗАЄМОДІЯ ПРИКЛАДНОГО РІВНЯ. ПРОТОКОЛ ПЕРЕДАЧІ ГІПЕРТЕКСТУ HTTP

2.1 Мета роботи

Вивчення методів та принципів роботи протоколу передачі гіпертексту HTTP. Отримання навичок розробки клієнт-серверних додатків, що взаємодіють на прикладному рівні, з використанням Internet-компонентів в середовищах програмування.

2.2 Теоретичні відомості для самостійної підготовки

2.2.1 Загальні відомості про протокол HTTP

HTTP (Hypertext Transfer Protocol, Протокол передачи гипертекста) представляет собой протокол прикладного уровня семейства протоколов TCP/IP. Описание протокола HTTP можно найти в RFC-1945, RFC-2068, RFC-2069, RFC-2616, RFC-2617. Протокол HTTP самый распространенный прикладной протокол стека TCP/IP. Когда вы посещаете различные WWW сайты с помощью браузера, браузер взаимодействует с WEB серверами, используя именно протокол HTTP. Всякий раз при переходе по гиперсвязи от одного ресурса к другому браузер обращается к HTTP для доступа к серверу, хранящему необходимую информацию. HTTP обеспечивает высокопроизводительный механизм тиражирования информации мультимедийных систем независимо от типа представления данных. Протокол построен по объектно- ориентированной технологии и может использоваться для решения различных задач, например, работы с серверами имен или управления распределенными информационными системами. На данный момент существует несколько версий протокола HTTP. Первой версией протокола HTTP была версия 0.9, в которой использовался простой запрос и поддерживался единственный метод GET. Таким образом, с помощью этого протокола можно было только получить документ или файл с сервера. До недавнего времени наиболее распространенной являлась версия 1.0. В 1996 году была стандартизирована версия 1.1, которая в настоящий момент поддерживается большинством серверов в сети Интернет. В этих версиях используется полный запрос, и набор методов значительно расширен. В частности можно не только получать файлы с сервера, но и передавать файлы на сервер, удалять их с сервера (естественно, при наличии определенных прав), передавать разными способами информацию специальным программам работающим на сервере, а также управлять параметрами соединения, кэширования, вида, типа и кодировки ресурса и т.д.

2.2.2 Сервіси WWW

Протокол HTTP позволяет получать доступ к ресурсам и сервисам WWW-серверов. Для унификации доступа к многофункциональным ресурсам сети, WWW-сервера поддерживают комплекс интерфейсов, позволяющих структурировать уровни и методы работы с различными ресурсами сети.

URI (Uniform Resource Identifier, Идентификатор ресурса), URL (Uniform Resource Locator, Местонахождение ресурса), URN (Uniform Resource Name, Имя ресурса) — разные имена одного и того же сервиса, который предназначен для идентификации типов, методов работы и компьютеров, на которых находятся определенные ресурсы, доступные через Internet. Этот сервис состоит из трех частей:

• Схема. Идентифицирует тип сервиса, через который можно получить доступ к сервису, например FTP или WWW-сервер

• Адрес. Идентифицирует адрес (хост) ресурса, например, www.nure.ua.

• Имя или путь доступа. Идентифицирует полный путь к ресурсу на выбранном хосте, который мы хотим использовать для доступа к ресурсу, например, /home/images/imagel.gif

Существует два типа адресов: абсолютный URI и относительный URI. Абсолютный URI содержит полный адрес объекта и протокол, т.е. все три вышеприведенные части. Например, файл logo.png, расположенный в каталоге images на сервере nure.ua, будет иметь следующий абсолютный URI

https://nure.ua/wp-content/themes/nure/images/nure_90_logo.png?v=2.0

Это означает, что будет использоваться тип доступа через HTTPS, схема доступа отделена двоеточием ":" и указывает на использование протокола HTTPS. Следующие два слэша отделяют последующий адрес сервера nure.ua, далее идет путь к файлу и сам файл. Абсолютный URI всегда должен начинаться со схемы доступа. Если путь к файлу и имя файла отсутствуют, то предполагается, что обращение происходит к корневому каталогу сервера "/". Следует отметить, что схема доступа и DNS адрес сервера являются регистро-независимыми, однако, путь и имя файла в общем случае регистро-зависимы.

Относительный URI использует URI текущего документа, запрошенного HTTP клиентом. Применяя ту же схему, HTTP клиент реконструирует URI, изменяя только некоторые имена и расширения файлов. Указание перед относительным URI “..“ значит переход на уровень каталогов вверх, а указание слеша – игнорирует все каталоги и добавляет указанный далее путь непосредственно к адресу сервера.

Кроме того, URI ресурса может содержать не только имя ресурса, но и параметры, необходимые для его работы. Имя ресурса отделено от строки параметров символом "?". Строка параметров состоит из лексем, разделяемых символом "&". Каждая такая лексема состоит из имени параметра и его значения, разделенных символом "=". Символы, не входящие в набор символов ASCII, заменяются знаком "%" и шестнадцатеричным значением этого символа. Например, символ "?" заменяется на "%3F". Поскольку символ пробела встречается в строке параметров довольно часто, то он заменяется не на «%20», а на символ “+” (если же в строке параметров встречается символ «+», то он заменяется на «%2B»). Для указанного ресурса вся строка параметров является одним строковым параметром, поэтому тип, очередность или уникальность имен отдельных параметров строки не существенны.

Документы на WWW-серверах обычно представлены в формате HTML. Hyper Text Markup Language (HTML) это язык разметки гипертекстов, который используется для представления информации в World Wide Web. HT в HTML обозначает Hyper Text, основную концепцию размещения информации в WWW. Hyper Text или hyperlinks (гиперссылки), содержит связи (URL) внутри текстового документа, которые позволяют пользователю быстро переходить от одной части документа к другой или к другому документу.

Common Gateway Interface (CGI) — это стандарт расширения функциональности WWW, позволяющий WWW-серверам выполнять программы. Причем пользователь может через WWW-сервер передавать различные параметры этим программам, задавая тем самым различные режимы их работы. Таким образом, CGI предоставляет возможность получения динамически сформированной информации с WWW-сервера. Механизм CGI полностью платформенно-независимый и позволяет CGI сервису обмениваться данными с любым WWW-сервером, поддерживающим механизм CGI. Поскольку CGI основан на исполняемых файлах, нет ограничений на типы программ, которые будут использовать интерфейс CGI. Программы могут быть разработаны на любом языке программирования, позволяющем создавать исполняемые модули. CGI-программы также могут быть написаны с использованием командных языков операционных систем, таких как Perl или Shell.

2.2.3 Принципи реалізації HTTP-з’єднання

Протокол HTTP построен по модели "запрос/ответ". Иными словами, клиент устанавливает соединение с сервером и отправляет запрос. В нем указан тип запроса, URL, версия протокола HTTP и содержание запроса: информация клиента (параметры) и, возможно, сопровождающая информация или тело сообщения. Сервер HTTP после обработки запроса возвращает ответ, который содержит: версию поддерживаемого протокола, код обработки запроса или код ошибки и возвращаемую по запросу информацию. Информация тела сообщений, как клиента, так и сервера должна быть представлена в MIME-формате.

HTTP-соединение инициируется клиентом и состоит из запроса к ресурсу определенного сервера. В простейшем случае, соединение представляет собой поток данных между клиентом — инициатором соединения и сервером. Однако довольно часто в соединении может участвовать промежуточный агент или Proxy сервер.

Proxy сервер это промежуточный агент, который принимает запрос клиента и, в зависимости от своих настроек, изменяет часть или все сообщение запроса и передает переформатированный запрос запрашиваемому серверу. В момент принятия запросов proxy может работать как сервер, а при передаче запросов — как клиент. Кроме того, proxy может поддерживать внутренний кэш запросов и ответов. Кэш хранит ответы серверов и возвращает их по запросу клиента, не передавая запрос непосредственно запрошенному серверу. Тем самым уменьшается время соединения, и увеличивается производительность работы с удаленными и медленными серверами. Однако далеко не все ответы могут кэшироваться. Некоторые запросы могут содержать параметры, накладывающие ограничения на работу кэша.

HTTP – это протокол прикладного уровня, который работает поверх транспортного протокола TCP. Однако, как всякий протокол прикладного уровня, может работать на любом другом транспортном протоколе, ориентированном на установку соединения и обеспечивающим гарантированную доставку. По умолчанию HTTP протокол использует TCP порт 80. В отличие от других прикладных протоколов, обеспечивающих неразрывное соединение до тех пор, пока не произойдет ошибка или не будет подан сигнал к завершению соединения, HTTP работает по-другому. HTTP-соединение должно открываться клиентом перед каждым запросом и закрываться сервером после отправки ответа. Ни браузер (клиент), ни сервер не хранят информацию даже о последнем соединении. Такой стиль работы позволяет серверу быстрее переходить к обслуживанию других клиентов, что увеличивает эффективность его работы. Однако при получении гипертекстовых документов, которые содержат встроенные графические объекты или другие ассоциированные объекты, за короткий промежуток времени браузер отправляет несколько запросов к одному и тому же серверу. В этом случае снижается эффективность работы и увеличивается загрузка сети за счет большого количества служебных TCP пакетов, предназначенных для открытия и закрытия соединения. Поэтому уже в спецификации HTTP/1.0 был предусмотрен режим неразрывного или постоянного соединения. В этом случае сервер не разрывает соединение после отправки ответа, и клиент может отправить следующий запрос. В спецификации HTTP/1.1 постоянные соедине ния используются по умолчанию. За выбор режима и управление параметрами соединения отвечает поле заголовка Connection. Кроме этого в HTTP/1.1 при использовании постоянных соединений может быть использована конвейерная обработка запросов. При этом клиент может отправить несколько запросов, не дожидаясь ответа на каждый, а затем получить несколько ответов от сервера. Следует также отметить, что даже при использовании постоянных соединений, сервер все же разрывает соединение с клиентом, если по истечении некоторого таймаута не получает от него никаких данных. Таймаут обычно составляет порядка нескольких десятков секунд. Поэтому, несмотря на возможность использования постоянных соединений, протокол HTTP все-таки отличается от других прикладных протоколов стека TCP/IP, в которых соединение обычно закрывается со стороны клиента.

2.2.4 Формат HTTP-транзакции.

Все HTTP-транзакции имеют один общий формат. Каждый запрос клиента и ответ сервера состоит из трех частей: первой строки запроса (ответа), заголовка и тела. Клиент инициирует транзакцию следующим образом:

- устанавливает TCP соединение с сервером по назначенному номеру порта (по умолчанию - 80);
- посылает первую строку запроса, указав HTTP-команду, называемую методом, адрес документа и номер версии HTTP;
- посылает заголовок запроса, чтобы сообщить серверу информацию о своей конфигурации и данные о форматах документов, которые он может принимать. Вся информация заголовка указывается построчно, при этом в каждой строке приводится имя и значение. Завершается заголовок пустой строкой;

- посылает тело запроса, которое для некоторых запросов может отсутствовать. Данные, находящиеся в теле, используются главным образом теми CGI-программами, которые применяют метод POST.

Сервер отвечает на запрос клиента следующим образом:

- посылает первую строку ответа, содержащую три поля: версию HTTP, код состояния и описание;
- посылает клиенту заголовок ответа, содержащий данные о самом сервере и затребованном документе. Как и в запросе заголовок завершается пустой строкой;
- посылает клиенту тело ответа. В случае успешного запроса тело содержит затребованные данные. Это может быть копия файла или результат выполнения CGI-программы. Если запрос клиента удовлетворить нельзя, тело может содержать дополнительные данные в виде понятного для пользователя разъяснения причин, по которым сервер не смог выполнить данный запрос;
- разрывает соединение с клиентом.

2.2.5 Структура та параметри запиту клієнта

Первая строка запроса представляется в виде:

METHOD <SP> Request-URI <SP>HTTP-Version

(здесь и далее <SP> обозначает символ пробела, /* line */ — обозначение пустой строки. Каждая строка заканчивается парой <CRLF> — возврат каретки, перевод строки.)

Поле "METHOD" содержит идентификатор метода обращения к HTTP-серверу. Метод определяет состав и структуру заголовков запроса, вид передачи и структуру параметров запроса. Наиболее распространенными методами являются "GET" и "POST". Иногда также используются методы "HEAD", "DELETE", "PUT". В спецификации HTTP1.1 добавлены некоторые новые методы, например "OPTIONS" и "TRACE". Все идентификаторы методов являются регистро-зависимыми.

"Request-URI" содержит абсолютный или относительный URI ресурс. В случае относительного URI схема доступа считается http, а в качестве адреса сервера берется, адрес сервера, с которым установлено соединение.

"HTTP-Version" указывает на версию HTTP поддерживаемую клиентом. По умолчанию используется версия 0.9. Для идентификации версии протокола HTTP использует схему "major.minor", т. е. версия HTTP/1.0 старше версии HTTP/0.9.

Например, первая строка запроса клиента может выглядеть следующим образом:

GET /index.html HTTP/1.0

Далее следует заголовок, который состоит из строк вида FieldName: FieldValue, где FieldName – название поля, FieldValue – значение поля. Каждая строка завершается символами <CRLF>, порядок следования полей не важен. Имена полей являются регистро-независимыми, значения полей могут быть регистро-зависимыми. Спецификация HTTP/1.0 требует, чтобы символ ":" следовал сразу за

именем поля, а между символом ":" и значением поля был только один символ <SP>. Спецификация HTTP/1.1 предъявляет менее жесткие требования. Условно можно разделить все поля запроса на следующие разделы: General-Header – основные поля, Request-Header – поля запроса, Response-Header – поля ответа, Entity-Header – поля объекта, Extension-Header – дополнительные поля. Наиболее распространенные поля для каждого раздела приведены ниже:

[General-Header]
Date
Pragma
Cache-Control
Connection
Trailer
Transfer-Encoding
Upgrade
Via

[Request-Header]
Authorization From
If-Modified-Since
Referer User-Agent Accept
Accept-Charset Accept-Encoding Accept-Language Expect
Host
If-Match
If-None-Match
If-Range
If-Unmodified-Since
Max-Forwards
Proxy-Authorization
Range
TE

[Response-Header]
Location
Server
WWW-Authenticate
Age
Proxy-Authenticate
Retry-After

Vary ETag Warning

[Entity-Header]
Allow

Content-Encoding Content-Length Content-Type Expires
Last-Modified Content-Language Content-Location Content-MD5
Content-Range

[Extension-Header]

Поля разделов "General-Header" и "Entity-Header" могут использоваться как в запросах клиента, так и в ответах сервера. Поля раздела "Request-Header" используются только в запросах клиента. Поля раздела "Response-Header" используются только при ответах сервера. В разделе "Extension-Header" содержатся некоторые дополнительные поля.

Заголовок запроса может выглядеть, например следующим образом:

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
*Accept: image/gif, text/html, */**
From: info@rubika.com.ua
If-Modified-Since: Sat, 26 Oct 2019 19:43:31 GMT
Connection: Keep-Alive

Заголовок отделяется от тела сообщения пустой строкой. Тело сообщения может присутствовать в запросе клиента при использовании методов "PUT" и "POST". При этом обязательно необходимо указать поле заголовка "Content-Length".

2.2.6 Структура та параметри відповіді серверу

Первая строка ответа представляется в виде

HTTP-Version <SP> Ret-Code <SP>Comment-String

Поле версии содержит номер версии HTTP, которую использует данный сервер для передачи ответа. "Ret-Code" это трехразрядное число, обозначаю- щее результат обработки сервером запроса клиента. Формат кода возврата аналогичен кодам возврата других прикладных протоколов стека TCP/IP, та- ких как FTP и SMTP. Первая цифра кода статуса HTTP определяет класс кода ответа, а каждый из пяти классов содержит группу значений статуса кода. Существует пять возможных значений для первой цифры (от 1 до 5):

1xx: информационные коды – запрос получен, продолжается обработка;

2xx: успешные коды – данные были успешно приняты, обработаны и использованы;

3xx: коды перенаправления – для выполнения запроса требуются дополнительные действия;

4xx: коды ошибок клиента – запрос содержит синтаксические ошибки либо запрошенные ресурсы не доступны;

5xx: коды ошибок сервера – сервер не смог обработать или выполнить запрос.

"Comment-String" - описание, представляющие собой просто понятный для пользователя текст, поясняющий код возврата.

Наиболее распространенные коды возврата приведены ниже:

100 – продолжать, Continue
200 – OK
201 – создан, Created
202 – принято, Accepted
204 - нет содержимого, No Content
206 - частичное содержимое, Partial Content
300 - множественный выбор, Multiple Choices
301 - постоянно перенесен, Moved Permanently
302 - временно перемещен, Moved Temporarily
304 - не модифицирован, Not Modified
305 - необходимо использовать прокси-сервер, Use Proxy
400 - некорректный Запрос, Bad Request
401 - несанкционированно, Unauthorized
403 - запрещено, Forbidden
404 - не найден, Not Found
405 - метод не дозволен, Method Not Allowed
407 - требуется аутентификация через прокси-сервер, Proxy Authentication Required
408 - истекло время ожидания запроса, Request Timeout
413 - объект запроса слишком большой, Request Entity Too Large
414 - URI запроса слишком длинный, Request-URI Too Long
500 - внутренняя ошибка сервера, Internal Server Error
501 - не реализовано, Not Implemented
505 - не поддерживаемая версия HTTP, HTTP Version Not Supported

Например, первая строка ответа сервера может выглядеть следующим образом:

HTTP/1.0 200 OK

или

HTTP/1.0 404 Not Found

Далее следует заголовок, аналогичный по формату заголовку запроса. Заголовок отделяется от тела сообщения пустой строкой. Размер тела сообщения указывается в поле заголовка "Content-Length". Допускается также в ответе не указывать поле Content-Length, в этом случае клиент может определить размер тела по закрытию соединения сервером. Кроме того, в спецификации HTTP/1.1 можно

при помощи поля Transfer-Encoding указать тип передачи "chunked". В этом случае размер тела сообщения указывается в самом теле, и поле Content-Length также может отсутствовать. Заголовок ответа может выглядеть, например следующим образом:

Date: Fri, 10 Jan 2019 08:17:58 GMT Server: Apache/1.2.6
Last-modified: Mon, 12 Jun 2015 21:53:08 GMT
Content-type: text/html
Content-length: 2482

2.2.7 Методи запитів

Метод "GET" запрашивает информацию о ресурсе, расположенном по заданному URI. Как правило, запрашиваемая информация представляет собой HTML, TXT или другой текстовый или графический файл. Если URI ассоциирован с исполняемым файлом — сервисом CGI, процесс CGI будет запущен, и клиенту будут переданы результаты стандартного потока вывода этого процесса. Тело запроса при использовании метода GET всегда пустое. Поэтому, если необходимо передать дополнительные данные, они присоединяются к URI в первой строке запроса. Если в теге <form> документа задано значение атрибута method="GET", то пары "ключ=значение", представляющие собой введенные данные из формы, присоединяются к URI после вопросительного знака (?). Пары отделяются друг от друга амперсантом (&). Символы, не входящие в набор символов ASCII, заменяются знаком "%" и шестнадцатеричным значением этого символа. Кроме того в методе GET может передаваться информация о дополнительных путях. При этом дополнительный путь указывается после URL (после имени файла и до знака вопроса), т.е. /cgi-bin/getaccount.exe/cgi/my_doc.txt?login=vasia. Сервер определяет, где заканчивается имя программы (getaccount.exe). Все данные, которые следуют за именем программы до знака вопроса, интерпретируются как дополнительный путь (этот путь будет доступен в cgi программе через переменную окружения PATH_INFO).

Запрос с помощью метода GET может выглядеть следующим образом:

GET /index.html HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
Host: nure.ua
*Accept: image/gif,image/jpeg, text/html, */**

Или в случае передачи данных CGI программе следующим:

GET /cgi-bin/getaccount.exe?user=peter&pass=pit&showdetail=1 HTTP/1.0
Host: www.ukr.net
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)

Метод "POST", как правило, используется для передачи клиентом на сервер данных, которые должны быть обработаны ресурсом, указанным в URI. Данный метод чаще всего используется для работы с CGI-сервисом. Метод "POST" передает параметры ресурсу URI в теле сообщения, поэтому при его использовании не требуется соблюдать никаких ограничений на длину передаваемой строки параметров. Если в теге <form> документа задано значение атрибута method="POST", то пары "ключ=значение", представляющие собой введенные данные из формы, помещаются в тело запроса. Размер тела запроса определяется полем заголовка "Content-Length", это поле является обязательным при использовании метода POST. Ответы на запросы методом POST, как правило, не кэшируются. Ниже приведен пример запроса с помощью метода POST.

POST /cgi-bin/getaccount.exe HTTP/1.1
Host: www.ukr.net
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98) Content-type: application/x-www-form-urlencoded
Content-Length: 32
/ line */*
user=peter
&pass=pit
&showdetail=1

Метод "HEAD" аналогичен методу "GET", за исключением того, что клиенту возвращается только заголовок сообщения ответа. Этот метод используется, когда клиент хочет получить информацию о документе, не получая сам документ. Для метода HEAD существует множество приложений. Например, клиент может затребовать следующую информацию:

- время изменения документа (эти данные полезны для запросов, связанных с кэш-памятью);
- размер документа (необходим для компоновки страницы, оценки времени передачи, определения необходимости запроса более компактной версии документа);
- тип документа (позволяет клиенту получать документы только определенного типа);
- тип сервера (позволяет создавать специализированные запросы).

Метод "DELETE" используется для удаления ресурса определенного URI. Для выполнения этого метода необходимо чтобы пользователь имел соответствующие права на удаляемый URI. Обычно эта возможность для пользователей не прошедших авторизацию на сервере запрещена. Запрос с помощью метода DELETE может выглядеть следующим образом:

DELETE /pub/user1/temp.txt HTTP/1.0
Connection: Keep-Alive

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)

Метод "PUT" используется, когда клиент желает сохранить передаваемый на сервер ресурс с идентификатором URI. Сам сохраняемый ресурс находится в теле запроса. Также как и для метода DELETE для выполнения этого метода пользователь должен иметь необходимые права на каталог куда помещается ресурс. Размер тела сообщения определяется полем заголовка "Content-Length", это поле является обязательным при использовании метода PUT. Ниже приведен пример запроса с помощью метода PUT.

PUT /pub/user1/upload/newpage.htm HTTP/1.0

Connection: Keep-Alive

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98) Content-type: text/html

Content-Length: 60

/ line */*

<HTML><BODY>My new Page is UNDER CONSTRUCTION

</BODY></HTML>

В спецификации HTTP/1.1 были добавлены несколько новых методов.

Метод "OPTIONS" запрашивает информацию о коммуникационных параметрах сервера. При этом указанный ресурс не запрашивается и, если это CGI-модуль, не запускается. Чтобы запросить данные обо всем сервере в целом, вместо URI запроса следует использовать символ '*'. Запрос с помощью метода OPTIONS может выглядеть следующим образом:

*OPTIONS * HTTP/1.1*

Host: www.ukr.net

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)

Метод "TRACE" применяется для организации локальной петли (loop-back) на прикладном уровне. HTTP-запрос при использовании метода TRACE не должен содержать тела. При получении такого запроса, сервер должен немедленно отправить поступившую информацию обратно в теле HTTP-ответа. Этот метод позволяет трассировать возможные соединения и получать информацию о временах прохождения и обработки запросов. Кроме того, проанализировав поле заголовка Via, клиент может узнать, через какие proxy сервера был пройден запрос. Поле заголовка Max-Forwards позволяет ограничивать длину цепочки запросов, что может быть полезным при тестировании бесконечных циклов в цепочке proxy серверов.

Кроме того, существует еще несколько дополнительных методов, которые поддерживаются не всеми серверами: CHECKIN, CHECKOUT, LINK, UNLINK, SEARCH, SHOWMETOD, SPACEJUMP, TEXTSEARCH и CONNECT. Метод CHECKOUT похож на GET, за исключением того, что он блокирует объект от изменений его другими пользователями, метод CHEKIN, напоминающий PUT, убирает блокировку. Используя метод LINK, можно связать существующий объект Web с другими объектами в сети. Для того чтобы убрать ссылку, используется метод UNLINK. Метод SHOWMETOD применяется для приёма информации о конкретном методе (методе, примененном к определённому объекту). Метод SPACEJUMP можно использовать для обработки запроса, указанного в терминах

координат в рамках объекта. Метод TEXTSEARCH применяется для запроса специфического объекта. Метод CONNECT используется proxy серверами, поддерживающими SSL туннелирование.

2.2.8 Поля заголовков запроса и ответа.

Общий заголовок "General Header". Этот заголовок может использоваться как в запросе, так и ответе сервера.

• Поле "Date:" содержит дату и время построения сообщения (запроса или ответа). Формат даты описан в RFC-822.

• Поле "Pragma:" используется для установки специальных директив участникам соединения. Например, опция "no-cache" не позволяет промежуточным объектам соединения использовать кэш при ответе на запрос.

В спецификации HTTP/1.1 в основной заголовок "General-Header" сообщения были добавлены следующие поля:

• Поле "Cache-Control:" позволяет более гибко управлять механизмами кэширования информации, устанавливать ограничения на время и объекты кэширования, управлять ограничениями на запросы из кэша определенных клиентов и др.

• Поле "Connection:" управляет параметрами соединения. В спецификации HTTP/1.0 для использования постоянного соединения клиент должен указать в этом поле значение "Keep-Alive". Как уже было отмечено ранее в спецификации HTTP/1.1 постоянные соединения используются по умолчанию. Поэтому клиенту HTTP/1.1 не обязательно указывать значение "Keep-Alive" в этом поле для использования постоянного соединения и в общем случае это поле может отсутствовать. Если же клиент, использующий HTTP/1.1, не желает устанавливать постоянное соединение, то он должен указать значение "Close" в поле Connection. В этом случае сервер разорвет соединение сразу после отправки ответа клиенту. В спецификации HTTP/1.1 поле Connection со значением "Close" может также указываться и в ответе. Таким образом, сервер сообщает клиенту, что соединение будет разорвано сразу после окончания передачи HTTP-ответа, и клиент не должен больше посылать запросы по этому соединению. Сервер при получении в заголовке запроса поля Connection со значением "Keep-Alive", может включить в заголовок ответа поле Keep-Alive. В этом поле он может указать параметры постоянного соединения. Например, время таймаута по истечении которого соединение будет разорвано, если от клиента не получено никаких данных.

• "Transfer-Encoding:" используется для указания типа кодировки тела запроса или ответа при передачи его по сети. В спецификации HTTP/1.1 в основном используется тип кодировки "chunked". В этом случае тело передается частями (chunk). Для каждой части в первой строке указывается размер части, затем содержимое, после которого следуют символы <CRLF>. Последняя часть имеет нулевой размер и может содержать дополнительные параметры, завершающиеся также символами <CRLF>. Таким образом, модно передавать тело любого размера, не указывая в заголовке поля Content-Length. Это может быть полезным, например, при динамической генерации HTTP-ответов.

• Поле "Upgrade:" содержит спецификации других протоколов, по которым может быть установлено соединение с данным сервером (или клиентом), помимо HTTP.

• Поле "Via:" используется для журнализации промежуточных Proxy-агентов или шлюзов, через которые прошел запрос (или ответ). Это поле подобно полю "Received:" в формате почтового сообщения.

Например, заголовок "General Header" может выглядеть следующим образом:

Date: Tue, 16 Nov 1999 08:12:31 GMT Pragma: no-cache
Connection: Close

Заголовок запроса "Request-Header" появляется только в запросах клиентов. Он позволяет клиентам отправлять на сервер дополнительную информацию о себе.

• Поле "Authorization:" содержит информацию для аутентификации пользователя. Ответ на запрос, содержащий поле "Authorization:" не кэшируется промежуточными объектами соединения.

• Поле "From:" может содержать email-адрес пользователя. Поле может использоваться для сбора различной статистики посещения Web сервера.

• Поле "If-Modified-Since:" используется при работе методом "GET". Если запрашиваемый ресурс не изменялся с момента, указанного в этом параметре, данный ресурс не возвращается, по запросу возвращается только заголовок сообщения ответа и соответствующий код возврата. Это поле может использоваться для контроля над содержимым кэша.

• Поле "Referer" содержит URI предшествующего ресурса. Этот параметр может использоваться сервером, например, для оптимизации кэширования, построения путей поиска и т. д.

• Поле "User-Agent:" содержит информацию о программном обеспечении клиента.

Заголовок запроса "Request-Header" был расширен новыми полями в спецификации HTTP/1.1:

• Поле "Accept:", предписывает отвечающему серверу использовать в ответе только указанные форматы данных. Например, если клиент понимает только текстовые данные и картинки в формате "gif", запрос должен содержать следующее:

Accept: text/, image/gif*

• Поля "Accept-Charset:", "Accept-Encoding:", "Accept-Language:" определяют, соответственно, допустимые в ответе значения символьного набора, типов кодировок и используемого национального языка. Кроме того, запрос может содержать поле "Content-Language:", описывающее основные (но не все) национальные языки, используемые при формировании тела сообщения.

• Поле "Host:" должно содержать IP или DNS адрес хоста у которого запрашивается URI ресурс. В общем случае через двоеточие может быть указан HTTP порт, используемый этим хостом (если порт не указан, то по умолчанию используется 80 порт). Это может понадобиться, например, для организации прозрачных (transpanent) proxy серверов. Обычно при использовании поля Host в

запросе указывается относительный URI, т.е. без адреса хоста, а адрес хоста указывается в поле Host (например, Internet Explorer всегда указывает в запросе относительный URI, за исключением случая, когда он настроен на работу непосредственно через proxy сервер). Таким образом, запрос может пройти через цепочку proxy серверов без указания каких-либо дополнительных полей и опций. Это поле является обязательным при использовании HTTP версии 1.1:

GET /examples/mainpage.htm HTTP/1.1
…….
Host: www.windows.edu:8000

• Поля "If-Match:", "If-None-Match:", "If-Range:", "If-Unmodified-Since:" устанавливают различные флаги на предыдущие запросы указанного ресурса для более эффективного управления кэшем сервера и/или промежуточных агентов.

• Поле "Max-Forwards:" содержит максимальное число промежуточных агентов, через которые может пройти запрос. Это поле может использоваться только вместе с методами TRACE и OPTIONS. Каждый промежуточный агент уменьшает это поле на единицу перед передачей запроса следующему агенту или конечному серверу. Если при приеме запроса это поле содержит ноль, то промежуточный агент не должен передавать запрос дальше, а должен сформировать HTTP-ответ для клиента.

• Поле "Proxy-Authorization:" позволяет клиенту регистрироваться на хостах промежуточных агентов соединения. Это поле может служить для аутентификации клиента на всем пути к запрашиваемому ресурсу.

• Поле "Range:" позволяет контролировать размер передаваемого сообщения. С помощью этого поля можно получить не весь документ целиком, а только некоторую его часть. Это позволяет получить запрашиваемый ресурс по частям или получить только оставшуюся часть ресурса при разрыве соединения. В качестве параметра для этого поля указывается единица измере- ния размера частей, а далее после символа "=" задается один или несколько диапазонов разделенных символом ",". Например bytes=0-19,100-149,-10 позволяет получить первые 20 байт, 50 байт, начиная с позиции 100, и последние 10 байт.

Например, заголовок "Request-Header" может выглядеть следующим образом:

Host: nure.ua
Authorization: Basic U2Fua292OlNlcmdleQ==
From: info@nure.ua
If-Modified-Since: Sat, 2 Sep 2019 19:43:31 GMT
Referer: http://nure.ua/home.html
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
*Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */* Accept-Language: ua*
Accept-Encoding: gzip, deflate
Range: bytes=0-99

Заголовок "Response-Header" позволяет передавать дополнительную информацию обработки запроса, которую нельзя поместить в строку статуса.

• Поле "Location:" содержит полный URI созданного или перемещенного ресурса. Например, если клиент отправил запрос на WWW-сервер с целью получения какого-либо документа, который был перемещен в другой каталог или на другой сервер, и запрашиваемый сервер отвечает "Location:...", то за- прос клиента автоматически перенаправляется по URI, указанному сервером.

• Поле "Server:" содержит спецификацию программного обеспечения WWW-сервера, отвечающего на запрос.

• Поле "WWW-Authenticate:" содержит параметры схемы и пространства аутентификации.

В спецификации HTTP/1.1 в основной заголовок "Response-Header" сообщения были добавлены следующие поля:

• Поле "Age:", которое содержит количество секунд, затраченное сервером на отправку ответа по данному запросу. Поле позволяет контролировать производительность различных серверов и промежуточных объектов сети.

• Поле "ETag:" содержит ключевые слова данного ресурса, по которым ресурс идентифицируется как на сервере, так и в кэше промежуточных аген- тов.

• Поле "Proxy-Authenticate:" содержит параметры схемы и пространства аутентификации для proxy сервера.

• Поле "Retry-After:" предписывает клиенту повторить запрос через указанное количество секунд или после указанной даты.

• Поле "Warning:" содержит дополнительную информацию о статусе обработки запроса — своеобразная система подчиненных кодов коду статуса ответа.

Например, заголовок "Response-Header" может выглядеть следующим образом:

Location: http://nure.ua/newsite/NewHome.htm
WWW-Authenticate: basic realm="NURE Security"
Server: Apache/1.2.6

Заголовок передаваемого сообщения "Entity-Header". Содержит информацию о структуре и формате тела запроса, или, если тело сообщения отсутствует, информацию о запрашиваемом ресурсе. Может появляться как в запросе клиента, так и ответе сервера.

• Поле "Allow:" содержит список методов, поддерживаемых ресурсом.

Это поле носит исключительно информационный характер.

• Поле "Content-Encoding:" содержит идентификатор типа дополнительной кодировки ресурса (т. е. для использования этого ресурса он должен быть декодирован указанным алгоритмом).

• Поле "Content-Length:" определяет длину тела сообщения.

• Поле "Content-Type:" содержит тип ресурса и таблицу кодировки представляемых данных (us-ascii, iso8580-l, windows-1251, koi8-r и др., в соответ- ствии со спецификацией MIME).

• Поле "Expires:" содержит дату окончания срока действия ресурса. Это поле, например, может использоваться в механизмах кэширования, как параметр запрета кэширования устаревших ресурсов. Это поле может также информировать пользователя о том, что информация, предоставляемая данным ресурсом, устарела.

• Поле "Last-Modified:" содержит дату и время последнего изменения ресурса.

В спецификации HTTP/1.1 раздел "Entity-Header" расширен полями:

• Поля семейства "Content*:", а именно: "Content-Base:", "Content-Language:", "Content-Location:", "Content-MD5:", "Content-Range:" – описывают, соответственно, формат, национальный алфавит, расположение частей ресурса (если таковые есть), алгоритм шифрования, размер частей и всего сообщения.

Например, заголовок "Entity-Header" может выглядеть следующим образом:

Allow: GET, HEAD Content-Encoding: x-gzip Content-Length: 3495

Content-Type: text/html; charset=windows-1251

Expires: Thu, 04 Sep 2001 16:00:00 GMT

Last-Modified: Mon, 04 Sep 2000 10:05:26 GMT

В разделе "Extension-Header" содержатся дополнительные поля, которые могут быть добавлены в спецификацию позднее. Все нераспознанные поля должны игнорироваться HTTP-клиентами, а proxy-сервера должны передавать их без изменений. Таким образом, функциональность HTTP протокола может быть расширена без изменения общей структуры запросов и ответов. Следует отметить, что некоторые из полей, добавленные в спецификации HTTP/1.1, для обеспечения совместимости включены как дополнительные поля в спецификацию HTTP/1.0. Одно из таких полей – это поле "Connection:", которое первоначально отсутствовало в спецификации HTTP/1.0. В спецификации HTTP/1.1 добавлены несколько полей, используемые для работы с proxy серверами. Например, поле X-Forwarder-For используется proxy серверами для указания IP адреса клиента, для которого выполняется запрос. Поле Proxy-Connection управляет параметрами соединения с proxy сервером.

2.2.9 Механізм Cookies

Как уже было отмечено, в общем случае для каждого запрашиваемого ресурса открывается новое соединение. При этом ни HTTP-клиент, ни HTTP-сервер не хранят информацию даже о последнем соединении. Поэтому если HTTP-клиент после запроса какого-либо ресурса повторно запрашивает с то го же сервера другой ресурс, то сервер не может определить тот же это клиент либо другой. В то же время для решения многих задач необходимо идентифицировать клиента на время сеанса работы пользователя, или даже между сеансами пользователя. Например, это может понадобиться для организации различных систем голосования и опроса пользователей, виртуальных магазинов, для аутентификации пользователей и сохранения различных настроек для каждого пользователя. Использование постоянного HTTP соединения в этом случае не решает проблему, поскольку такое

соединение предназначено для последовательного запроса нескольких документов. Постоянное соединение не может оставаться активным на время всего сеанса работы пользователя, так как оно будет разорвано сервером по истечению сравнительно небольшого таймаута, если от клиента не поступает никаких данных.

Для решения этой проблемы протокола HTTP разработан специальный механизм cookies. В этом случае информацию о соединении должен хранить HTTP-клиент, например браузер. Для реализации этого механизма используются два дополнительных поля HTTP-заголовка Set-Cookie и Cookie. Схема взаимодействия клиента и сервера при использовании механизма cookies, выглядит следующим образом:

- клиент запрашивает какой-либо документ с сервера, например, обращается к CGI-модулю;
- сервер включает в заголовок ответа поле Set-Cookie, в котором сообщает клиенту информацию, предназначенную для сохранения, и параметры, задающие область действия этой информации;
- при каждом следующем запросе клиент возвращает серверу сохраненную информацию с помощью поля Cookie в заголовке запроса, если запрашиваемый ресурс попадает в область действия, заданной сервером;
- CGI-модуль или другой ресурс на сервере может проанализировать значение поля Cookie и идентифицировать клиента.

Первоначально механизм cookies был описан в спецификации "Netscape Communications Persistent Client State HTTP Cookies", в дальнейшем был принят RFC-2109 "HTTP State Management Mechanism", описывающий этот механизм. Общий принцип работы и формат полей заголовка у этих спецификаций совпадают, однако они имеет небольшие различия в параметрах заголовков.

В спецификации RFC-2109 поле Set-Cookie выглядит следующим образом:

Set-Cookie: NAME=VALUE; Comment=comment; Domain=domain;
Max-Age=delta-seconds; Path=path; Secure; Version=1

Обязательными параметрами являются NAME и Version. RFC-2109 описывает версию 1, если параметр Version опущен, то используется спецификация netscape. В этой спецификации вместо параметра Max-Age используется параметр Expires и отсутствует параметр Comment. Остальные параметры идентичны в обеих спецификациях. Параметр NAME задает сами данные, которые должны быть сохранены клиентом. Параметр Domain задает имя домена, для которого действительно значение переданного cookie. По умолчанию используется доменное имя сервера, который сформировал ответ. Параметр Path задает каталог на сервере, для документов которого действительно значение переданного cookie. Если указан "Path=/", то cookie действительны для всего сервера. Если параметр опущен, то используется каталог, из которого был запрошен документ. Если присутствует поле Secure, то cookie передаются только по защищенному HTTPS соединению. Параметры Max-Age и Ex- pires задают срок действия cookie. При этом Max-Age содержит количество секунд, по истечению которых переданный cookie считается недействительным и должен быть удален браузером. А поле Expires содержит дату, после наступления которой переданный cookie считается недействительным. Если поля опущены, то cookie действительны только на время одного сеанса пользователя

(т.е. пока окно браузера не будет закрыто), и браузер не должен со- хранять значение cookie на диске. Поле Set-Cookie не должно кэшироваться proxy серверами.

Рассмотрим пример взаимодействия клиента и сервера при использовании механизма cookie.

1. HTTP-клиент ⇒ HTTP-сервер
POST /forum/cgi-bin/login.cgi HTTP/1.1
Host: nure.ua
… … …
user=serg&pass=secret

2. HTTP- сервер ⇒ HTTP- клиент
HTTP/1.1 200 OK
Set-Cookie: user=serg; Version=1; Path=/forum
… … …

3. HTTP-клиент ⇒ HTTP-сервер
POST /forum/cgi-bin/sendmes.cgi HTTP/1.1
Host: nure.ua
Cookie: $Version=1; user=serg; $Path=/forum
… … …

Таким образом, cookie представляет собой всего лишь блок данных, хранящийся на стороне клиента. Формирование и анализ этих данных должны выполняться на стороне сервера, например с помощью CGI-модулей. При сохранении cookie браузер должен придерживаться следующих ограничений:

- всего может храниться до 300 значений cookies
- каждый cookie не может превышать 4Кбайт
- с одного сервера или домена может храниться до 20 значений cookie.

Следует отметить, что механизм cookies нельзя назвать надежным механизмом идентификации пользователей. Поскольку браузер может удалить сохраненные cookies, например при нехватке свободного места, либо вообще не поддерживать механизма cookies. Кроме того, пользователь может запретить в браузере поддержку cookies, или удалить сохраненные на диске cookies вручную.

2.3 Розробка клієнт-серверного додатку за протоколом HTTP

2.3.1 Internet API

Для разработки клиентских HTTP приложений, работающих под управлением операционных систем семейства Windows, можно использовать специальные Internet API функции. Эти функции предоставлены в рамках WinINet Software Development Kit, и реализованы в виде динамической библиотеки wininet.dll. Полное описание всех функций Internet API находится в ActiveX Software Development Kit

(SDK), раздел Win32 Internet Programmer's Reference или в Win32 SDK. Все функции можно разделить на четыре группы: общие URL функции, HTTP - функции, FTP - функции и Gopher функции. Для разработки клиентского HTTP приложения можно использовать функции первых двух групп. Первая группа функций позволяет работать с любым ресурсом, заданным через URL, как с обычным файлом. То есть можно получить дескриптор ресурса с помощью функции InternetOpenUrl и дальше производить чтение и запись с помощью функций InternetReadFile и InternetWriteFile. Причем всю работу по формированию HTTP запросов и обработки HTTP ответов возьмет на себя интерфейс Internet API. Перед использованием указанных выше функций необходимо сначала открыть Internet соединение с помощью функции InternetOpen. После окончания работы с соединением его необходимо закрыть, используя функцию InternetCloseHandle. Эта функция используется также для освобождения всех дескрипторов созданных другими функциями Internet API. Функции второй группы явля- ются специализированными для протокола HTTP. Например, функция HTTPSendRequest позволяет отправлять на сервер HTTP запрос.

Еще одним способом разработки клиентских HTTP приложений, работающих под управлением операционных систем семейства Windows, является использование специальных классов и программных компонент, предоставляемых системами визуального программирования. Эти классы и компоненты скрывают особенности работы протокола HTTP и предоставляют прикладному программисту набор методов и данных для удобной работы по протоколу HTTP. Одним из наиболее распространенных подобных компонентов является TNMHTTP. Рассмотрим принципы работы этого компонента более подробно.

2.3.2 Принципы работы компонента TNMHTTP

Он является одним из группы компонентов, предназначенных для работы с прикладными протоколами стека протоколов TCP/IP, разработанных фирмой NetMaster. TNMHTTP является потомком компонента TPowerSock и наследует от него ряд свойств методов и событий, характерных для всей группы компонентов TNM*. Рассмотрим основные из них.

Свойства.

property Host: String; содержит IP или DNS адрес сервера, с которым будет установлено соединение.

property Port: Integer; содержит номер порта сервера. Для HTTP по умолчанию используется 80 порт. property LocalIP: string;

property RemoteIP: string; содержат соответственно адрес локального компьютера и адрес сервера после установки соединения.

property ReportLevel: integer; степень детальности сообщений, возвращаемых в событии OnStatus. Может содержать значения:

Status_None = 0,

Status_Informational = 1,

Status_Basic = 2,

Status_Routines = 4,

Status_Debug = 8,

Status_Trace = 16.

property Status: String; последнее значение строки статуса, которое было возвращено в событии OnStatus.

property LastErrorNo: integer; последний код ошибки интерфейса сокетов. property Handle: tSocket; дескриптор сокета, используемого интерфейсом сокетов.

property TimeOut: Integer; время ожидания ответа от сервера в миллисекундах, прежде чем будет сгенерирована исключительная ситуация. Если значение этого свойства установлено в 0, то таймаут не используется и ожидание будет бесконечным.

События.

property OnStatus: TOnStatus; возникает при изменении состояния сокета. Текущий статус передается в обработчик события через параметр Status типа String.

Если адрес сервера задан в виде DNS, то перед соединением необходимо сначала определить IP адрес сервера. В результате выполнения этой операции может возникнуть два события.

property OnHostResolved: TOnHostResolved; возникает при успешном определении IP адреса.

property OnInvalidHost: THandlerEvent; возникает, если IP адрес определить не удалось. В этом случае можно задать новый адрес сервера и установить параметр handled в true в обработчике этого события. После чего будет выполнена повторная попытка определения IP адреса и подключения к серверу. Обработчик этого события может выглядеть следующим образом:

```
void    fastcall TForm1::NMHTTP1InvalidHost(bool &handled)

{ AnsiString NewHost;
if (InputQuery("Invalid Host", "Please Choose another host", NewHost))
{NMHTTP1->Host = NewHost;
handled = true;
}
}
```

property OnConnect: TNotifyEvent; возникает при успешном установлении соединения с сервером.

property OnConnectionFailed: TNotifyEvent; возникает, если соединение с сервером установить не удалось.

property OnDisconnect: TNotifyEvent; возникает при разрыве соединения с сервером.

Методы. TpowerSock имеет достаточно много методов для приема и передачи данных, однако в компоненте TNMHTTP добавляются свои специализированные методы для выполнения основных методов протокола HTTP. Из методов TpowerSock можно особо выделить метод function Transaction(const CommandString: String): String; virtual;

Он передает серверу строку CommandString и возвращает ответ сервера. Кроме того, ответ также запоминается в свойстве property TransactionReply: String; а свойство property ReplyNumber: Smallint; содержит числовой код результата выполнения транзакции. При использовании TNMHTTP это свойство содержит цифровой код ответа HTTP сервера.

Помимо унаследованных от TpowerSocket свойств, методов и событий TNHHTTP имеет свои свойства, методы и события специализированные именно для работы с протоколом HTTP. Основными здесь являются методы, названия которых совпадают с методами запросов HTTP протокола. Наиболее часто используются следующие методы:

procedure Get(URL: string); virtual; procedure Head(URL: string); virtual; procedure Post(URL, PostData: string); virtual; procedure Delete(URL: string); virtual; procedure Put(URL, PutData: string); virtual; procedure Options(URL: string); virtual;

Во всех методах параметр URL указывает адрес запрашиваемого ресурса. Допускается указание как полного абсолютного адреса, например http://www.nure.ua:8080/main.htm, так и относительного адреса, например /images/image1.jpg. Причем при указании полного абсолютного адреса, включающего схему доступа, свойства Host и Port игнорируются, а адрес сервера и порт для подключения берутся из URL. По умолчанию используется 80 номер порта. При указании абсолютного URL без схемы доступа свойство Host игнорируется, однако свойство Port, если оно задано, учитывается. При задании относительного URL, который должен начинаться с символа слеша "/", учитываются оба свойства Host и Port. Кроме этого поддерживается возможность работы с сервером через прокси-сервер, для чего используются свойства Proxy и ProxyPort.

property Proxy: String; содержит DNS или IP адрес прокси-сервера

property ProxyPort: integer; содержит порт, на котором работает прокси-сервер.

При отправке запроса серверу имеется возможность заполнить некоторые поля заголовка с помощью свойства HeaderInfo.

property HeaderInfo: THeaderInfo;

Объект THeaderInfo включает в себе следующие свойства:

property Cookie: string; содержит значение cookies, посылаемых серверу. property LocalMailAddress: string; содержит e-mail адрес клиента, соответствует полю From: заголовка.

property LocalProgram: string; идентификация программы HTTP клиента, соответствует полю User-Agent заголовка.

property Referer: string; содержит адрес документа, где содержалась ссылка на данный документ, соответствует одноименному полю заголовка.

property UserId: string; и property Password: string; содержат соответственно имя пользователя и пароль, при использовании аутенфикации в режиме Basic.

Перед отправкой запроса все поля заголовка собираются в строку доступную в свойстве property SendHeader: TExStringList;

Непосредственно перед отправкой запроса возникает событие OnAbout-ToSend, в обработчике которого можно добавить дополнительные поля к за головку

запроса. Например, обработчик этого события может выглядеть следующим образом:

```
void  fastcall TForm1::NMHTTP1AboutToSend(TObject *Sender)
{
NMHTTP1->SendHeader->Insert(NMHTTP1->SendHeader->Count-1, "If-Modified-Since: Wed, 01 Jun 2002 00:25:00 GMT");
}
```

Для методов Post и Put тело запроса задается в параметрах PostData и PutData соответственно. Причем если свойство OutputFileMode установлено в значение true, то эти параметры должны содержать имена файлов на локальном компьютере, из которых будет считываться информация для отправки. Если же свойство OutputFileMode установлено в значение false, то эти поля должны содержать сами данные для отправки.

После отправки запроса на сервер ожидается ответ от сервера. Числовой код ответа записывается в свойство ReplyNumber. Само содержимое HTTP ответа может быть получено с помощью свойств property Body: string; и property Header: string; Причем если свойство InputFileMode установлено в значение true, то эти свойства перед выполнением HTTP запроса должны содержать имена файлов на локальном компьютере, куда будут помещены принятые тело и заголовок HTTP ответа. Если же свойство InputFileMode установлено в значение false, то после ответа сервера в эти свойства будут непосредственно помещены тело и заголовок HTTP ответа. Кроме того, в свойстве CookieIn содержится значение cookies, переданных сервером. Следует отметить, что все методы, выполняющие HTTP запросы, например Get, возвращают управление только после ответа сервера. Однако внутри этих методов просматривается очередь сообщений вызвавшего их потока и поэтому приложение не будет блокировано на время выполнения этих методов.

Для прерывания выполнения методов можно использовать метод Abort. Причем иногда при получении некоторых некорректных ответов от сервера, например, при получении пустого ответа, без вызова этого метода приложение не сможет корректно завершить работу. Таким образом, для отправки запроса GET можно использовать следующий код (пример с использованием proxy):

```
NMHTTP1->TimeOut = 10000; NMHTTP1->InputFileMode = false; NMHTTP1->OutputFileMode = false; NMHTTP1->ReportLevel = Status_Basic;
if (CheckBox1->Checked) //Включен режим работы через прокси сервер
{ NMHTTP1->Proxy = "proxy.edu" NMHTTP1->ProxyPort = 3128;
}
NMHTTP1->HeaderInfo->LocalMailAddress = "user@proxy.edu"; NMHTTP1->HeaderInfo->LocalProgram = "Simple HTTP client";
if (CheckBox2->Checked) //Используется аутентификация в режиме Basic
{ NMHTTP1->HeaderInfo->UserId = "gst"; NMHTTP1->HeaderInfo->Password = "gst";
```

```
}
try
{
NMHTTP1->Get("http://www.nure.ua/docs/index.htm"); Memo1->Text = NMHTTP1->Body;
Memo2->Text = NMHTTP1->Header;
if (NMHTTP1->CookieIn != "") ShowMessage("Cookie:\n"+NMHTTP1->CookieIn);
}
catch(ESockError &E)
{
MessageBox(Handle,E.Message.c_str(),"Connection Error!!!",MB_OK|MB_ICONERROR);
}
```

После выполнения запроса возникает событие OnSuccess, если запрос был выполнен успешно, и событие OnFailure, если возникли какие-либо ошибки при выполнении запроса. Параметр Cmd в обработчике этих событий содержат код запроса, для которого они произошли.

2.4 Порядок виконання лабораторної роботи

1) Вивчити основні методи HTTP версії 1.1 - GET, HEAD, POST, PUT, DELETE, TRACE та OPTIONS, а також – основні поля заголовків та відповідей протоколу HTTP і кодів відповіді серверу.

2) Вивчити структуру HTTP запитів та HTTP відповідей.

3) Розробити клієнт-серверний додаток відповідно до варианту завдання (див. Додаток 2.1).

4) Скласти звіт про виконання лабораторної роботи. Звіт та клієнт-серверний додаток завантажити у відповідний каталог Гугл-диску.

5) Захистити виконану работу.

2.5 Зміст звіту

Звіт про виконання лабораторної роботи повинен повністю відображати хід виконання лабораторної роботи та відповідати вимогам методичних вказівок до кожної лабораторної роботи. Звіт про виконання лабораторної роботи повинен містити:

1) титульний аркуш;

2) мету роботи;

3) індивідуальне завдання на лабораторну роботу;

4) хід виконання роботи з описом отриманих результатів. Всі важливі моменти та хід виконання індивідуального завдання повинні бути проілюстровані у вигляді копій екранів з отриманими результатами;

5) висновки;

6) відповіді на контрольні запитання.

Робочі проекти (схеми) оформлюються як додатки до звіту про виконання лабораторної роботи.

Звіт про виконання лабораторної роботи подається викладачеві або в електронному вигляді (якщо це оговорено викладачем), або на аркушах формату А4, скріплених між собою. Про відпрацювання лабораторної роботи та допуск до її захисту свідчить підпис викладача на титульному аркуші. Для захисту студент повинен представити підписаний звіт та усі робочі документи з виконання лабораторної роботи (файли звіту, проекту, додатки тощо).

2.6 Контрольні запитання та завдання

1. Призначення протоколу HTTP та загальні принципи роботи протоколу.
2. Формат URI-адрес ресурсів.
3. Формат HTML-документів.
4. Інтерфейс CGI.
5. Загальна схема взаємодії клієнта та сервера при HTTP-з'єднанні.
4. Проаналізувати формат HTTP-запита та пояснити призначення його частин.
5. Проаналізувати формат HTTP-відповіді та пояснити призначення його частин.
6. Основні методи запита в специфікації HTTP/1.1, пояснити їх прзначення та принципи роботи кожного з методів.
7. Описати розділи заголовку, що використовуються в HTTP-запиті, та їх основні поля, пояснити їх призначення.
8. Описати розділи заголовку, що використовуються в HTTP-відповіді, та їх основні поля, пояснити їх призначення.
9. Принципи розробки клієнт-серверних додатків, що взаємодіють за протоколом HTTP, для операційних систем сімейства Windows.
10. Принципи роботи компонентів TNMHTTP, TNMURL та TNMUUProcessor.

Література

1. Ю.А. Семенов Протоколы Internet - М. : Горячая линия-Телеком, 2001.– 1100 с.
2. Фролов А.В., Фролов Г.В. Глобальные сети компьютеров. Практическое введение в Internet, E-Mail, FTP, WWW и HTML, программирование для Windows Sockets Том 23, М.: Диалог-МИФИ, 1996, 283 с.

Додаток 2.1

Варіанти завдань

Во всех вариантах необходимо реализовать HTTP-клиент с помощью компонента TNMHTTP. Пользователю должна быть предоставлена возможность задания IP или DNS адреса сервера, номера порта, а также путь и имя начального файла. В каждом варианте (кроме вариантов 9,10,11) HTTP-клиент должен получить с сервера указанный начальный файл и после этого выполнить действия согласно варианту задания. Для проверки работы программы можно в качестве начального файла указывать файл mainpage.htm, расположенный в каталоге examples на сервере. Кроме того, HTTP-клиент должен отображать в строке статуса текущее состояние соединения, а также тип последней операции и последний код возврата сервера.

1) HTTP-клиент должен получать все html-файлы на которые имеются ссылки в полученном файле (тег <a href="*.htm") и сохранять их на диске.

2) HTTP-клиент должен получать все графические файлы формата jpg на которые имеются ссылки в полученном файле (тег <img src="*.jpg") и сохранять их на диске.

3) HTTP-клиент должен получить фоновый звуковой файл (тег <bgsound src="*.mid") и фоновый графический файл (тег <body background="*.gif | *.jpg") на которые имеются ссылки в полученном файле и сохранить их на диске.

4) HTTP-клиент должен запрашивать информацию о трех первых html-файлах на которые имеются ссылки в полученном файле (тег <a href="*.htm") и получать самый маленький из этих трех файлов.

5) HTTP-клиент должен запрашивать информацию о трех первых графических файлах формата jpg на которые имеются ссылки в полученном файле (тег <img src="*.jpg") и получать самый большой из этих трех файлов.

6) HTTP-клиент должен запросить информацию о фоновом звуковом файле (тег <bgsound src="*.mid") и фоновом графическом файле (тег <body background="*.gif | *.jpg") на которые имеются ссылки в полученном файле и, в случае, если их общий размер не превышает указанного, получить эти файлы.

7) HTTP-клиент должен получить информацию о всех графических файлах формата gif на которые имеются ссылки в полученном файле (тег <img src="*.gif" и <body background="*.gif") и получить только те из них, раз- мер которых не превышает заданный.

8) HTTP- клиент должен вывести информацию обо всех ресурсах на которые имеются ссылки в полученном файле (тег <img src="*.*" и <a href="*.*"). Для каждого ресурса должен быть выведен тип ресурса, его размер и дата создания или последней модификации документа.

9) HTTP-клиент должен с помощью метода GET передать информацию скрипту (/cgi-bin/script.exe) на сервере, а затем отобразить документ возвращенный этим скриптом. Необходимо также на основе заголовка полученного ответа выводить версию и название программного обеспечения сервера, версию HTTP, поддерживаемую сервером, и размер полученного тела сообщения.

10) HTTP-клиент должен с помощью метода POST передать информацию скрипту (/cgi-bin/script.exe) на сервере, а затем отобразить документ возвращенный этим скриптом. Необходимо также на основе заголовка полученного ответа выводить размер и тип полученного документа, дату его последней модификации и версию HTTP, поддерживаемую сервером.

11) HTTP-клиент должен запросить и вывести содержимое указанного виртуального каталога на сервере, для просмотра которого сервер требует прохождение аутентификации в режиме Basic. (для проверки можно использовать каталог /secure на сервере и имя пользователя и пароль gst)