МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Брянский государственный технический университет

Утверждаю

Ректор университета

_____ О.Н. Федонин

«_____» _____ 2019г.

СЕТИ ЭВМ И ТЕЛЕКОММУНИКАЦИИ

РАЗРАБОТКА ПРОСТОГО HTTP-СЕРВЕРА С ИСПОЛЬЗОВАНИЕМ СОКЕТОВ

Методические указания
к выполнению лабораторной работы №14
для студентов очной формы обучения
по направлению подготовки
09.03.01 «Информатика и вычислительная техника»,
профиль «Программное обеспечение вычислительной техники и автоматизированных систем»

Брянск 2019

УДК 004.01

Сети ЭВМ и телекоммуникации. Разработка простого HTTP-сервера с использованием сокетов [Электронный ресурс]: методические указания к выполнению лабораторной работы №14 для студентов очной формы обучения по направлению подготовки 09.03.01 «Информатика и вычислительная техника», профиль «Программное обеспечение вычислительной техники и автоматизированных систем» – Брянск, 2019. – 20с.

Разработали:
А.О. Трубаков,
канд. техн. наук, доц.;
А.А. Трубакова,
асс.

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол № 4 от 24.12.2018г.)

Методические указания публикуются в авторской редакции

1. ЦЕЛЬ РАБОТЫ

Цель работы – изучить механизм межсетевого взаимодействия с серверной стороны на примере реализации протокола *HTTP*. Для этого студент должен:

- изучить основы программирования межсетевого взаимодействия с серверной стороны;
- углублено изучить протокол *HTTP*;
- научиться использовать протокол *HTTP* для разработки простого *HTTP*-сервера.

Продолжительность работы – 2 часа.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Понятие сервера и их типы

Сервер – это вычислительное устройство, обеспечивающее выполнение какой-либо задачи (программы) или ряда программ для некоторого количества персональных компьютеров. Исходя из этого, от его работы напрямую зависит работа всех пользователей, связанных с ним.

Ниже описываются некоторые распространенные типы серверов, классифицируемых по классу решаемых задач.

Веб-сервер (HTTP сервер) – сервер, принимающий *HTTP*-запросы от клиентов, обычно веб-браузеров, и выдающий им *HTTP*-ответы, как правило, вместе с *HTML*-страницей, изображением, файлом, медиа-потоком или другими данными. Веб-сервер обязан обеспечить бесперебойную работу Интернет-ресурса с учетом посещаемости, противостоять сетевым атакам, не допускать возможности взлома. Чем большую роль играет Интернет-сайт в бизнес-процессе (например, обеспечивает связь с клиентами, является каналом сбыта продукции), тем важнее для нее этот сервер. В последние годы веб-сервером называют чаще не саму машину, а программу, выполняющую вышеперечисленные функции.

Сервер приложений. Для сервера приложений характерны расширенные возможности обработки информации, а взаимодействие с клиентом становится подобным работе приложения. В маркетинге термином «сервер приложений» обычно обозначают предлагаемое продавцами комплексное решение, которое содержит все требуемые компоненты технологий. Для некоторых организаций такой комплексный подход к построению сервера приложений облегчает разработку благодаря унификации разрабатываемых моделей и централизации поддержки.

Сервер баз данных. Предназначены не столько для хранения и доступа, сколько для обработки массивов информации. Через клиентские запросы запрашиваемая информация извлекается, данные обрабатываются, структурируются, изменяются в зависимости от настроек сервера. Руководят работой таких серверов СУБД (Системы Управления Базами Данных), самые известные из них – *MS SQL Server, Oracle, MySQL*. В зависимости от количества пользователей и размера базы данных, а также перспективы их увеличения в будущем, определяют такие важные характеристики сервера базы данных, как мощность и масштабируемость.

Файл-сервер – это централизованное хранилище информации, доступ к дискам которого имеют подключенные в локальную сеть персональные компьютеры. Основная задача файлового сервера сводится к надежному сохранению данных и бесперебойному доступу к ней, а в случае повреждения файлов – полному их восстановлению.

Прокси-сервер – посредник между пользователями локальной сети и Интернетом. Обеспечивает безопасный выход в интернет, защищая от нежелательного доступа извне и при необходимости ограничивая выход на определенные ресурсы пользователям локальной сети. Кроме того, выполняет ряд других функций: учет и экономия трафика путем сжатия данных, кэширование, анонимизация доступа.

Почтовый сервер. Одна из задач почтового сервера – чтение адресов входящих сообщений и доставка корреспонденции в соответствующие почтовые ящики в пределах интрасети. В зависимости от развитости

почтового сервера он может предоставлять администратору большую или меньшую степень контроля над локальными почтовыми ящиками, типами и размерами сообщений, которые они в состоянии получать, автоматическими ответами, которые можно составлять.

Сервер DHCP. Используется для автоматического присваивания *IP*-адрес каждой локальной машине. Основное преимущество сервера *DHCP* – свобода изменения конфигурации локальной сети при ее расширении, добавлении или удалении машин (например, портативных ПК).

Принт-сервер. Позволяет использовать одно печатающее устройство для обслуживания нескольких компьютеров. Функции принт-сервера – принять запросы на вывод печати, выстроить их в очередь и согласно ей отправлять на принтер. Таким образом, экономятся средства на комплектацию каждого компьютера собственным принтером, их память освобождается для других задач, рационально используется офисное пространство.

Сервер удаленного доступа. Эти системы позволяют связываться с удаленной сетью по телефонным линиям. При наличии хороших каналов связи разница между работой в офисе и вне его в этом случае практически незаметна.

2.2. HTTP-сервер

Веб-сервер – сервер, принимающий *HTTP*-запросы от клиентов и выдающий им *HTTP*-ответы. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает.

Из всего серверного программного обеспечения, как считается, самыми распространенными являются *Apache* и *Microsoft IIS*. Первый является более популярным и в большей степени используется в *unix*-подобных системах, хотя и может устанавливаться в среду *Windows*. Кроме того, сервер *Apache* является абсолютно бесплатным программным обеспечением и совместим практически со всеми известными операционными системами.

Программный продукт от *Microsoft* рассчитан на среднестатистического пользователя, который установить и настроить такой веб-сервер для *Windows* сможет без дополнительной помощи квалифицированного специалиста.

2.3. Общий принцип работы HTTP-сервера

В работе *HTTP*-сервера необходимо различать сам сервер и клиент. В качестве клиентов для обращения к веб-серверам могут использоваться различные программы и устройства:

- веб-браузер, работающий на настольном компьютере или переносном устройстве (например, карманном ПК);
- разнообразные программы, самостоятельно обращающиеся к веб-серверам для получения обновлений или другой информации (например, антивирус может периодически запрашивать у определённого веб-сервера обновления своих баз данных);
- мобильный телефон, получающий доступ к ресурсам веб-сервера;
- другие цифровые устройства или бытовая техника.

Общий принцип работы заключается в том, что клиент передаёт веб-серверу запросы на получение ресурсов, обозначенных *URL*-адресами. Схематично обмен показан на рис. 1.

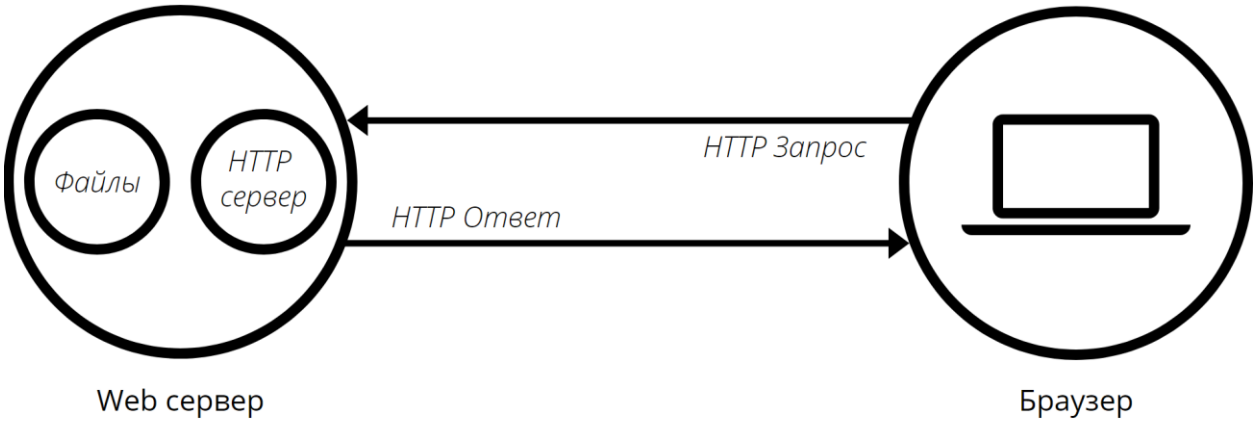

Рис. 1. Принцип работы сервера

Ресурсы – это *HTML*-страницы, изображения, файлы, медиа-потоки или другие данные, которые необходимы клиенту. В ответ веб-сервер передаёт

клиенту запрошенные данные (если он их не находит, то сообщает об ошибке 404). Этот обмен происходит по протоколу *HTTP*. Общий алгоритм работы *HTTP*-сервера представлен на рис. 2.

Рис. 2. Принцип работы HTTP-сервера

После того, как пользователь обратился к определенному ресурсу по протоколу *HTTP*, клиент (обычно браузер) формирует *HTTP*-запрос к веб-серверу. Обычно указывается символическое имя сервера (например, «*http://www.microsoft.com*») – в этом случае браузер предварительно преобразует это имя в *IP*-адрес при помощи сервисов *DNS*. После этого по протоколу *HTTP* на веб-сервер отправляется сформированное *HTTP*-сообщение. В этом сообщении браузер указывает какой ресурс необходимо загрузить и всю дополнительную информацию. Задача веб-сервера – прослушивать определенный *TCP*-порт (обычно порт 80) и принимать все входящие *HTTP*-сообщения. Если входящие данные не соответствуют формату сообщения *HTTP*, то такой запрос игнорируется, а клиенту возвращается сообщение об ошибке.

В простейшем случае при поступлении *HTTP*-запроса веб-сервер должен считать содержимое запрашиваемого файла с жесткого диска, упаковать его содержимое в *HTTP*-ответ и отправить клиенту. В случае если требуемый файл не найден на жестком диске, то веб-сервер сгенерирует ошибку с указанием статусного кода 404 и отправит это сообщение клиенту.

Такой вариант работы веб-сервера принято называть статическими сайтами. В этом случае на стороне сервера не запускается никакой программный код, кроме программного кода самого веб-сервера. Однако подобные сценарии работы все чаще оказываются непригодными, а им на смену приходят полноценные веб-приложения. Отличие таких приложений состоит в том, что *HTML*-документы и другие ресурсы не хранятся на сервере в виде неизменяемых данных. Вместо этого, на сервере хранится программный код, который способен сгенерировать эти данные в момент обработки запроса. Разумеется, некоторые ресурсы (такие как файлы каскадных стилей, изображения и т.д.) могут храниться как статическое содержимое, но основные страницы *HTML* генерируют в процессе обработки. В таком случае веб-сервер при обработке запроса *HTTP* должен обращаться к программному коду, который должен сгенерировать содержимое.

2.4. Протокол HTTP

2.4.1. Общие сведения

HTTP – широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (т.е. документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Аббревиатура *HTTP* расшифровывается как *HyperText Transfer Protocol*, «протокол передачи гипертекста». В соответствии со спецификацией *OSI*, *HTTP* является протоколом прикладного (верхнего, 7-го) уровня. Актуальная на данный момент версия протокола, *HTTP* 1.1, описана в спецификации *RFC* 1945, RFC 2616.

2.4.2. HTTP-транзакция

При обращении браузера к Веб-странице, располагаемой на удалённом узле, его клиентское программное обеспечение (например, *MS Internet Explorer*) запрашивает все документы, такие, как файл *HTML*, изображения и мультимедиа, из которых страница, собственно, и состоит. Процесс запроса и доставки этих файлов, регулируемый протоколом передачи гипертекста (*Hypertext Transport Protocol, HTTP*) – называется транзакцией *HTTP*.

Основным и единственным способом общения в *HTTP* протоколе является *HTTP* сообщение. Сообщения делятся на два вида: *HTTP*-запрос – это сообщение, которое клиент посылает серверу и *HTTP*-ответ – это сообщение, которое сервер посылает клиенту.

2.4.3. HTTP-запрос

HTTP запрос состоит из трех основных частей, которые идут в нем именно в том порядке, который указан ниже.

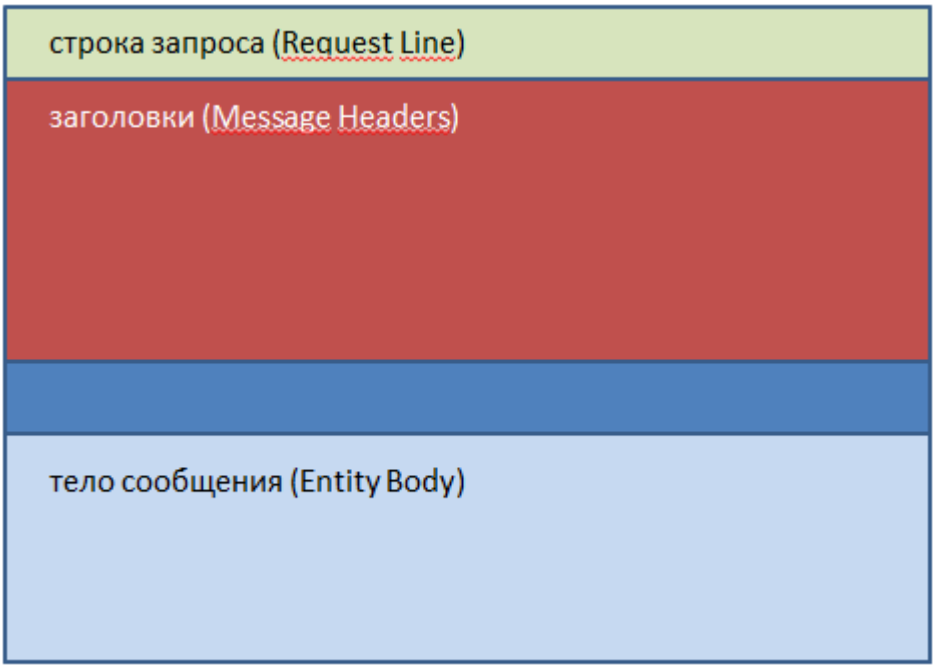

Рис. 3. HTTP-запрос

Запрос состоит из следующих частей:

1. ***Строка запроса*** – указывает метод передачи, *URL*-адрес, к которому нужно обратиться и версию протокола *HTTP*.

2. ***Заголовки*** – описывают тело сообщений, передают различные параметры и другие сведения и информацию.
3. ***Разделитель*** – между заголовками и телом сообщения находится пустая строка, которая представляет собой символ перевода строки.
4. ***Тело сообщения*** – это сами данные, которые передаются в запросе. Тело сообщения – это необязательный параметр и может отсутствовать.

Когда мы получаем ответный запрос от сервера, тело сообщения, чаще всего представляет собой содержимое веб-страницы. Но, при запросах к серверу, оно тоже может иногда присутствовать, например, когда мы передаем данные, которые заполнили в форме обратной связи на сервер.

Строка запроса выглядит так:

```
Метод URI HTTP/Версия
```

Параметры запроса:

- *Метод (англ. Method)* – тип запроса, одно слово заглавными буквами. В версии *HTTP* 0.9 использовался только метод *GET*.
- *URI* определяет путь к запрашиваемому документу.
- *Версия (англ. Version)* – пара разделённых точкой цифр. Например: 1.0.

Пример запроса:

```
GET /wiki/HTTP HTTP/1.0
Host: ru.wikipedia.org
```

Типы запросов:

- ***GET***. Используется для запроса содержимого указанного ресурса. С помощью метода *GET* можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.

Кроме обычного метода *GET*, различают ещё:

- условный *GET* – содержит заголовки *If-Modified-Since, If-Match, If-Range* и подобные;

- частичный *GET* – содержит в запросе *Range*.

Порядок выполнения подобных запросов определён стандартами отдельно.

- ***HEAD***. Аналогичен методу *GET*, за исключением того, что в ответе сервера отсутствует тело. Запрос *HEAD* обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация *URL*) и чтобы узнать, не изменился ли он с момента последнего обращения.

Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше — копия ресурса помечается как устаревшая.

- ***POST***. Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в *HTML*-форму, после чего они передаются серверу методом *POST* и он помещает их на страницу. При этом передаваемые данные (в примере с блогами – текст комментария) включаются в тело запроса. Аналогично с помощью метода *POST* обычно загружаются файлы на сервер.

В отличие от метода *GET*, метод *POST* не считается идемпотентным, то есть многократное повторение одних и тех же запросов *POST* может возвращать разные результаты (например, после каждой отправки комментария будет появляться очередная копия этого комментария).

При результате выполнения 200 (*Ok*) в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ 201 (*Created*) с указанием *URI* нового ресурса в заголовке *Location*.

Сообщение ответа сервера на выполнение метода *POST* не кэшируется.

2.4.4. HTTP-ответ

HTTP-ответ имеет аналогичную с запросом структуру. Стартовая строка ответа сервера имеет следующий формат:

```
HTTP/Версия КодСостояния Пояснение
```

В строке ответа используются следующие обозначения:

- *Версия* – пара разделённых точкой цифр, как в запросе.
- *КодСостояния (англ. Status Code)* – три цифры (первая из которых указывает на класс состояния), которые определяют результат совершения запроса. Например, в случае, если был использован метод *GET*, и сервер предоставляет ресурс с указанным идентификатором, то такое состояние задаётся с помощью кода 200. Если сервер сообщает о том, что такого ресурса не существует – 404. Если сервер сообщает о том, что не может предоставить доступ к данному ресурсу по причине отсутствия необходимых привилегий у клиента, то используется код 403. Спецификация *HTTP* 1.1 определяет 40 различных кодов *HTTP*, а также допускается расширение протокола и использование дополнительных кодов состояний.
- *Пояснение (англ. Reason Phrase)* – текстовое короткое пояснение к коду ответа для пользователя. Пояснение может не учитываться клиентским программным обеспечением, а также может отличаться от стандартного в некоторых реализациях серверного ПО.

После стартовой строки следуют заголовки, а также тело ответа. Например:

```
HTTP/1.1 200 OK
Server: nginx/1.2.1
Date: Sat, 08 Mar 2019 22:53:46 GMT
Content-Type: application/octet-stream
Content-Length: 5
Last-Modified: Sat, 08 Mar 2019 22:53:30 GMT
Connection: keep-alive
Accept-Ranges: bytes

Hello
```

Тело ответа следует через два переноса строки после последнего заголовка. Для определения окончания тела ответа используется значение

заголовка *Content-Length* (в данном случае ответ содержит 5 восьмеричных байтов: слово «*Hello*»).

2.4.5. Коды состояния

Клиент узнаёт по коду ответа о результатах его запроса и определяет, какие действия ему предпринимать дальше. Набор кодов состояния является стандартом, и они описаны в соответствующих документах *RFC*. Введение новых кодов должно производиться только после согласования с *IETF*. Клиент может не знать все коды состояния, но он обязан отреагировать в соответствии с классом кода.

В настоящее время выделено пять классов кодов состояния.

- ***Информационные (1xx).*** В этот класс выделены коды, информирующие о процессе передачи. При работе через протокол версии 1.0 сообщения с такими кодами должны игнорироваться. В версии 1.1 клиент должен быть готов принять этот класс сообщений как обычный ответ, но серверу отправлять что-либо не нужно. Сами сообщения от сервера содержат только стартовую строку ответа и, если требуется, несколько специфичных для ответа полей заголовка. Прокси-сервера подобные сообщения должны отправлять дальше от сервера к клиенту.
- ***Успех (2xx)***. Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения.
- ***Перенаправление (3xx)***. Коды этого класса сообщают клиенту, что для успешного выполнения операции необходимо сделать другой запрос, как правило, по другому *URI*. Из данного класса пять кодов 301, 302, 303, 305 и 307 относятся непосредственно к перенаправлениям. Адрес, по которому клиенту следует произвести запрос, сервер указывает в заголовке *Location*. При этом допускается использование фрагментов в целевом *URI*.

- ***Ошибка клиента (4xx)***. Этот класс кодов предназначен для указания ошибок со стороны клиента. При использовании всех методов, кроме *HEAD*, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя.
- ***Ошибка сервера (5xx)***. Эти коды выделены под случаи неудачного выполнения операции по вине сервера. Для всех ситуаций, кроме использования метода *HEAD*, сервер должен включать в тело сообщения объяснение, которое клиент отобразит пользователю.

3. ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

В процессе выполнения лабораторной работы студенту необходимо разработать простой *HTTP*-сервер, который сможет работать по протоколу *HTTP* 1.0. В лабораторной работе необходимо реализовать базовую часть и индивидуальное задание. К базовой части относиться поддержка всего одной команды *GET*. Если в качестве запрашиваемого параметра указан файл, который присутствует в папке программы, то он должен возвращаться клиенту. Иначе – необходимо сформировать ответ с ошибкой 404 (страница не найдена).

Если в качестве запроса на сервер была направлена другая команда, или команда с неправильным форматом, необходимо возвращать ошибку 501 (не реализовано).

При выполнении лабораторной работы желательно реализовать многопоточный сервер. Это позволит более углубленно освоить реализацию серверной части межсетевого взаимодействия. Однако допускается и создание однопоточного сервера.

Дополнительно к базовой функциональности необходимо выполнить индивидуальное задание, которое выбирается согласно варианту студента в журнале.

1. Добавить поддержку вывода списка файлов в папке. Если в качестве параметра команды *GET* задана папка, находящаяся в той же папке что и само приложение, то необходимо вернуть список файлов в этой папке.

2. Добавить поддержку перенаправления (статус ответа 303). Если был запрошен адрес /*iipo*, то сервер должен перенаправить пользователя на сайт iipo.tu-bryansk.ru.

3. Добавить ограничения на доступ к папке /*secret*. Если пользователь запросил данный адрес, то в любом случае возвращать ему статус 401 (требуется авторизация). При вводе авторизации все равно возвращать статус 401 (как если бы пароль был неверным).

4. Добавить эмуляцию исключительной ситуации, при которой сервер не может выполнить команду из-за превышения отведенного интервала времени. При запросе адреса /*timeout* сервер должен вернуть статус 408 (время ожидания истекло).

5. Реализовать корректную поддержку форматов (*Content-Type*) для наиболее популярных типов данных *MIME*. Определение и выбор типа производить по расширению запрошенного файла.

6. Добавить в ответ сервера поддержку дополнительных полей *Server*, *Date*, *Accept*.

7. Добавить ограничение клиентов. При запуске сервера необходимо предусмотреть ввод названия клиента, которым разрешено подключаться к серверу. Всем остальным клиентам сервер должен отвечать страницей с ошибкой 400 (плохой запрос).

8. Добавить поддержку куки (*cookie*). При любом обращении клиента на сервер, сервер формирует ответ, который будет устанавливать куку с именем author и значением, равным фамилии студента.

9. Эмулировать временную ошибку сервера. Если клиент запросил страницу /error, то необходимо вернуть статус 503 (временно не доступен).

10. Добавить ограничение на запрашиваемые ресурсы. Если клиент запросил файл с расширением *pdf*, то возвращать статус 415 (неподдерживаемый тип данных).

11. Добавить эмуляцию ошибки сервера. Если клиент запросил любой адрес с расширением *php*, то возвращать статус 500 (внутренняя ошибка сервера).

12. Добавить поддержку статуса 410 (ресурс удален). Если клиент запрашивает любой ресурс, который начинается с /*secret*/, то необходимо ему вернуть статус 410.

13. Добавить поддержку временного перенаправления (статус ответа 307). Если был запрошен адрес /*bstu*, то сервер должен перенаправить пользователя на сайт *www.tu-bryansk.ru*.

14. Эмуляция ошибки конфликта одновременного блокирующего использования некоторого ресурса. Если клиент обратился по адресу, начинающемуся с /*conflict*/, необходимо вернуть ему ошибку со статусом 409 (конфликт обращения).

15. Добавить поддержку проверки длинны запроса. Если клиент запрашивает страницу, в адресе которой больше 50 символов, то возвращать ошибку 414 (запрашиваемый *URL* слишком длинный).

16. Реализовать эмуляцию ошибки, которая возникает при отправке слишком большого запроса. Если клиент обращается по адресу, который начинается /*big*/, то необходимо вернуть ему ошибку со статусом 413 (размер запроса слишком велик).

17. Реализовать эмуляцию ошибки, которая возникает при исчерпывании сетевого трафика. Если пользователь запросил файл с расширением *mp3* или *avi*, то возвращать ему статус 509 (исчерпана пропускная ширина канала).

18. Если в запросе клиента присутствует параметр *info=yes*, то сервер не должен возвращать страницу с ответом, а должен вернуть страницу с описанием всех заголовков, переданных ему в запросе.

19. Добавить ограничения на доступ к папке /*forbidden*. Если пользователь запросил данный адрес, то возвращать ему статус 403 (доступ запрещен).

20. Добавить особую реакцию на запросы, в которых клиент запрашивает файл с расширением *.*exe*. В этом случае клиенту необходимо возвращать страницу, в которой будет написано, что доступ к подобным файлам запрещен.

21. Добавить особую реакцию на запрос клиента /*cookie*. При получении этого запроса сервер должен перечислить все куки, которые клиент передал ему в запросе.

22. Реализовать эмуляцию ошибки переполнения. Если клиентом запрошена страница /*error*, то необходимо вернуть статус 507 (переполнение).

23. Формировать правильный ответ клиенту только в том случае, если он передал поле *Accept-Language* значение *ru*. Если этого значения в списке доступных языков нет, то выдавать страницу с сообщением о том, что сервер поддерживает только русскоязычных клиентов.

24. Добавить ограничение на закачку архивов. Если клиент запрашивает файл с расширением *rar* или *zip*, то сервер должен вернуть статус 415 (неподдерживаемый тип данных).

25. Реализовать поддержку статуса 418 из RFC 2324. Данный статус должен появляться при обращении по адресу /*coffee* или /*tea*.

26. Если клиент пересылает в запросе на сервер куку с именем *request* и значением *info*, то сервер должен вместо ответа сформировать страницу, в которой он сообщит информацию о компьютере, на котором он запущен (например, его имя).

27. Реализовать поддержку многосайтовости. Если клиент в параметре host указал название server2, то выдавать ему результат из папки server2, иначе – из папки server1.

28. Реализовать поддержку команды *HEAD*.

29. Добавить поддержку статуса ответа 102. Если была запрошена страница по адресу /102, то необходимо возвращать периодически клиенту статус 102 (идет обработка), тем самым не давая клиенту разорвать подключение.

30. Добавить поддержку команды *OPTIONS* (определения параметров и возможностей сервера).

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

Для проверки своих знаний при подготовке к сдаче лабораторной работы можно использовать следующие контрольные вопросы.

1. Для чего используется протокол *HTTP*?
2. Для чего предназначены команды *GET, HEAD, PUT*, описанные в протоколе *HTTP* 1.0?
3. Что означают статусы ответа 200, 404, 500, переданные сервером в ответ на запрос клиента?
4. Чем отличается версия протокола *HTTP* 1.0 от версии 1.1?
5. Что такое *MIME*?
6. Для чего служит поле *host* в запросе клиента? Является ли оно обязательным или опциональным?
7. Что должно находиться в поле *content-type*, отправляемом в ответе сервера?
8. Что должно находиться в поле *content-length*, отправляемом в ответе сервера?
9. Что такое *cookie*? Может ли сервер менять их?

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

1. Куроуз, Д.Ф. Компьютерные сети. Нисходящий подход / Д.Ф. Куроуз, К.В. Росс. – М.: Эксмо, 2016. – 912 с.

2. Олифер, В.Г. Компьютерные сети. Принципы, технологии, протоколы: учеб. для вузов / В.Г. Олифер, Н.А. Олифер.– СПб: Питер, 2010. – 944с.

3. Олифер, В.Г. Сетевые операционные системы: учеб. для вузов / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2009. – 672 с.

4. Таненбаум, Э.С. Компьютерные сети. / Э.С. Таненбаум. – СПб.: Питер, 2009.– 992 с.

5. Берлин, А.Н. Телекоммуникационные сети и устройства / А.Н. Берлин. – Москва: ИНТУИТ, 2016. – 395 с. – Режим доступа: http://www.iprbookshop.ru/52197.html. – ЭБС «IPRbooks».

6. Буцык, С.В. Вычислительные системы, сети и телекоммуникации / С.В. Буцык, А.С. Крестников, А.А. Рузаков. – Челябинск: Челябинский государственный институт культуры, 2016. –116 с. – Режим доступа: http://www.iprbookshop.ru/56399.html. – ЭБС «IPRbooks».

7. Галас, В.П. Вычислительные системы, сети и телекоммуникации. Часть 1. Вычислительные системы / В.П. Галас. – Владимир: ВГУ им. А.Г. и Н.Г. Столетовых, 2016. – 232 с. – Режим доступа: http://www.iprbookshop.ru/57363.html. – ЭБС «IPRbooks».

8. Глухоедов, А.В. Инфокоммуникационные системы и сети. Конспект лекций / А.В. Глухоедов. – Белгород: Белгородский государственный технологический университет им. В.Г. Шухова, 2015. – 160 с. – Режим доступа: http://www.iprbookshop.ru/66654.html. – ЭБС «IPRbooks».

9. Платунова, С.М. Администрирование сети Windows Server 2012 / С.М. Платунова. – Спб: Университет ИТМО, 2015. –102 с. – Режим доступа: http://www.iprbookshop.ru/65769.html. – ЭБС «IPRbooks».

Сети ЭВМ и телекоммуникации. Разработка простого HTTP-сервера с использованием сокетов: методические указания к выполнению лабораторной работы №14 для студентов очной формы обучения по направлению подготовки 09.03.01 «Информатика и вычислительная техника», профиль «Программное обеспечение вычислительной техники и автоматизированных систем».

ТРУБАКОВ АНДРЕЙ ОЛЕГОВИЧ
ТРУБАКОВА АННА АЛЕКСЕЕВНА

Научный редактор Д.А. Коростелев
Компьютерный набор А.О. Трубаков
Иллюстрации А.О. Трубаков

Подписано в печать __.__.____. Усл.печ.л. 1,25 Уч.-изд.л. 1,25

Брянский государственный технический университет
241035, Брянск, бульвар 50 лет Октября, 7 БГТУ
Кафедра «Информатика и программное обеспечение», тел. 56-09-84

Сопроводительный лист на издание в авторской редакции

Название работы Сети ЭВМ и телекоммуникации. Разработка простого HTTP-сервера с использованием сокетов: методические указания к выполнению лабораторной работы №14 для студентов очной формы обучения по направлению подготовки 09.03.01 – «Информатика и вычислительная техника», профиль «Программное обеспечение вычислительной техники и автоматизированных систем»

Актуальность и соответствующий научно-методический уровень подтверждаю _____

(подпись научного редактора)

Рукопись сверена и проверена автором _____

(подпись автора)

Рекомендуется к изданию _____

(подпись заведующего кафедрой)