МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»

(БГТУ им. В.Г.Шухова)

**Лабораторная работа №8**

дисциплина «Сети ЭВМ и телекоммуникации»

тема: «Программирование протокола HTTP»

Выполнил: студент группы ВТ-31

Ковалёв И. Д.

Проверил: Федотов Е. А.

Белгород 2020

**Цель работы**: изучить протокол HTTP и составить программу согласно заданию.

**Теоретические сведения**

HTTP – протокол прикладного уровня стека протоколов TCP/IP, предназначенный для передачи данных по сети с использованием транспортного протокола TCP.

Протокол HTTP может использоваться также в качестве «транспорта» для других протоколов прикладного уровня, таких как SOAP или XML-RPC.

Основой HTTP является технология «клиент-сервер». HTTP-клиенты отсылают HTTP-запросы, которые содержат метод, обозначающий потребность клиента. Также такие запросы содержат универсальный идентификатор ресурса, указывающий на желаемый ресурс. Обычно такими ресурсами являются хранящиеся на сервере файлы. По умолчанию HTTP-запросы передаются на порт 80. HTTP-сервер отсылает коды состояния, сообщая, успешно ли выполнен HTTP-запрос или же нет.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное.

Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т.д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым. Унифицированный идентификатор ресурса представляет собой сочетание унифицированного указателя ресурса и унифицированного имени ресурса. Метод протокола HTTP – это команда, передаваемая HTTP-клиентом HTTP-серверу.

Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Название метода чувствительно к регистру. Каждый сервер обязан поддерживать как минимум методы GET и HEAD. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501. Если серверу метод известен, но он неприменим к конкретному ресурсу, то возвращается сообщение с кодом 405. В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

Код состояния HTTP представляет собой целое число из трех цифр. Первая цифра указывает на класс состояния:

· информационные сообщения;

· успешное выполнение;

· переадресация;

· ошибка клиента;

· ошибка сервера.

Каждое HTTP-сообщение состоит из трех частей, которые передаются в следующем порядке:

1. Стартовая строка – определяет тип сообщения;

2. Заголовки – характеризуют тело сообщения, параметры передачи и прочие сведения;

3. Тело сообщения – непосредственно данные сообщения.

Стартовые строки HTTP-сообщения различаются для запроса и ответа. Стартовая строка HTTP-запроса имеет следующий формат:

Метод URI HTTP/Версия, где метод - название запроса, URI определяет путь к запрашиваемому документу, версия - пара разделённых точкой арабских цифр.

Стартовая строка HTTP-ответа имеет следующий формат: HTTP/Версия Код Состояния Пояснение.

Заголовок HTTP представляет собой строку в HTTP-сообщении, содержащую разделенную двоеточием пару параметр-значение.

Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой. Все заголовки разделяются на четыре основных группы:

1. Основные заголовки – должны включаться в любое сообщение клиента и сервера.

2. Заголовки запроса – используются только в запросах клиента.

3. Заголовки ответа – только для ответов от сервера.

4. Заголовки сущности – сопровождают каждую сущность сообщения.

Тело HTTP-сообщения, если оно присутствует, используется для передачи данных, связанных с запросом или ответом.

Протокол HTTP устанавливает отдельную TCP-сессию на каждый запрос; в более поздних версиях HTTP было разрешено делать несколько запросов в ходе одной TCP-сессии, но браузеры обычно запрашивают только страницу и включённые в неё объекты (картинки, каскадные стили и т. п.), а затем сразу разрывают TCP-сессию. Для поддержки авторизованного (неанонимного) доступа в HTTP используются cookies; причём такой способ авторизации позволяет сохранить сессию даже после перезагрузки клиента и сервера.

При доступе к данным по FTP или по файловым протоколам тип файла (точнее, тип содержащихся в нём данных) определяется по расширению имени файла, что не всегда удобно. HTTP перед тем, как передать сами данные, передаёт заголовок «Content-Type: тип/подтип», позволяющую клиенту однозначно определить, каким образом обрабатывать присланные данные. Это особенно важно при работе с CGI-скриптами, когда расширение имени файла указывает не на тип присылаемых клиенту данных, а на необходимость запуска данного файла на сервере и отправки клиенту результатов работы программы, записанной в этом файле.

Кроме того, HTTP позволяет клиенту прислать на сервер параметры, которые будут переданы запускаемому CGI-скрипту. Для этого же в HTML были введены формы.

.

**Задание к работе**

1. Разработать программу, позволяющую принимать запрос на выдачу страницы от интернет-браузера и формировать ответ в зависимости от запроса. Реализовать методы GET и HEAD на стороне сервера.

2. Передать браузеру в сообщении тестовую страницу (определяется программистом) запрошенную страницу, или код ошибки если страница не найдена.

3. Программы должны быть написаны на языке программирования Pascal или C.

**Выполнение работы**

Программа была протестирована с помощью браузера и программы Advanced REST client. Переходя по адресу 127.0.0.1, сервер отдает файл index.html с таким содержимым:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Получилось</title>

</head>

<body>

<p>Что получилось?</p>

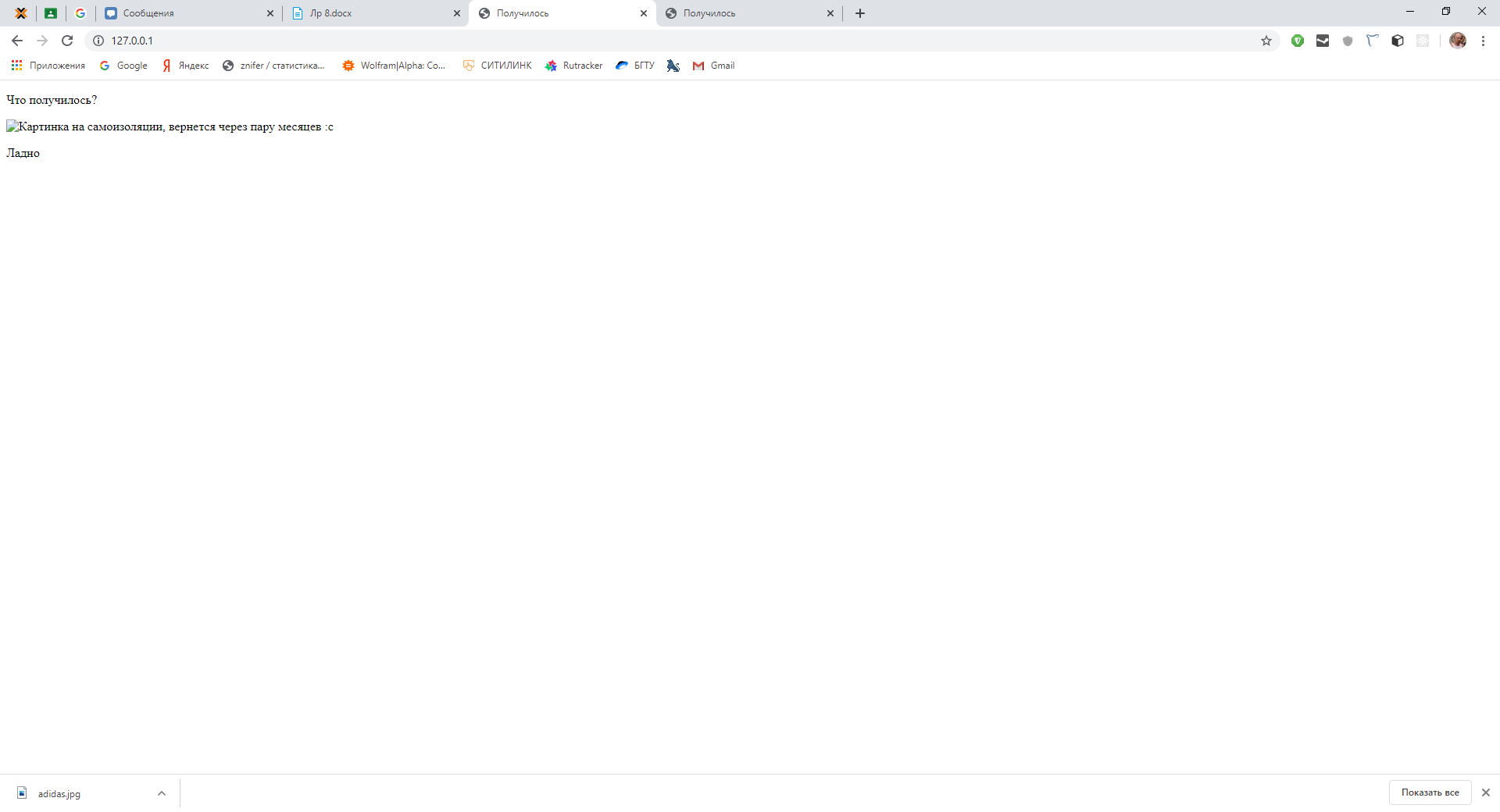
<img src="adidas.jpg" alt="Картинка на самоизоляции, вернется через пару месяцев :с">

<p>Ладно</p>

</body>

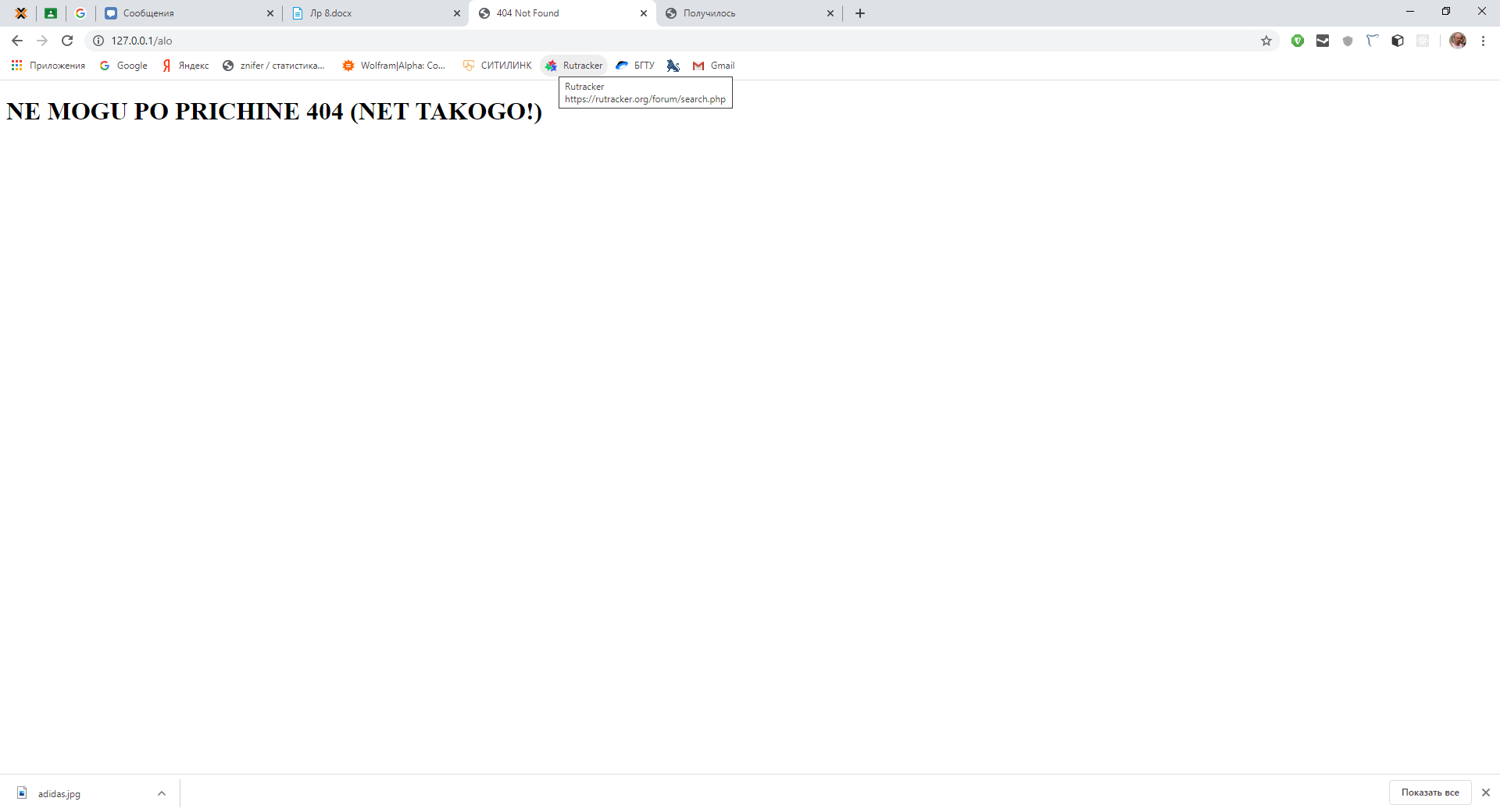
</html>

Проверим с помощью браузера:

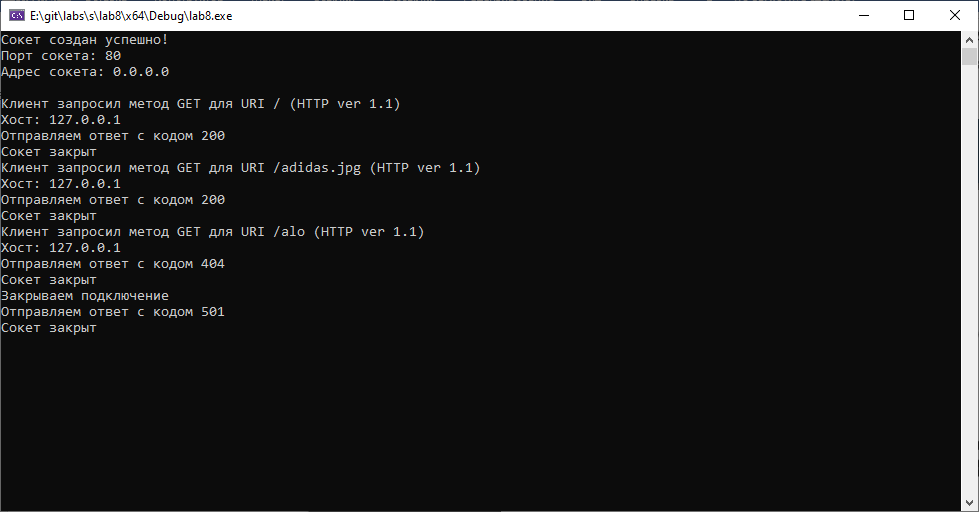


В случае, если серверу будет отправлен запрос на страницу, которой не существует, он должен отдать заглушку с ошибкой 404.

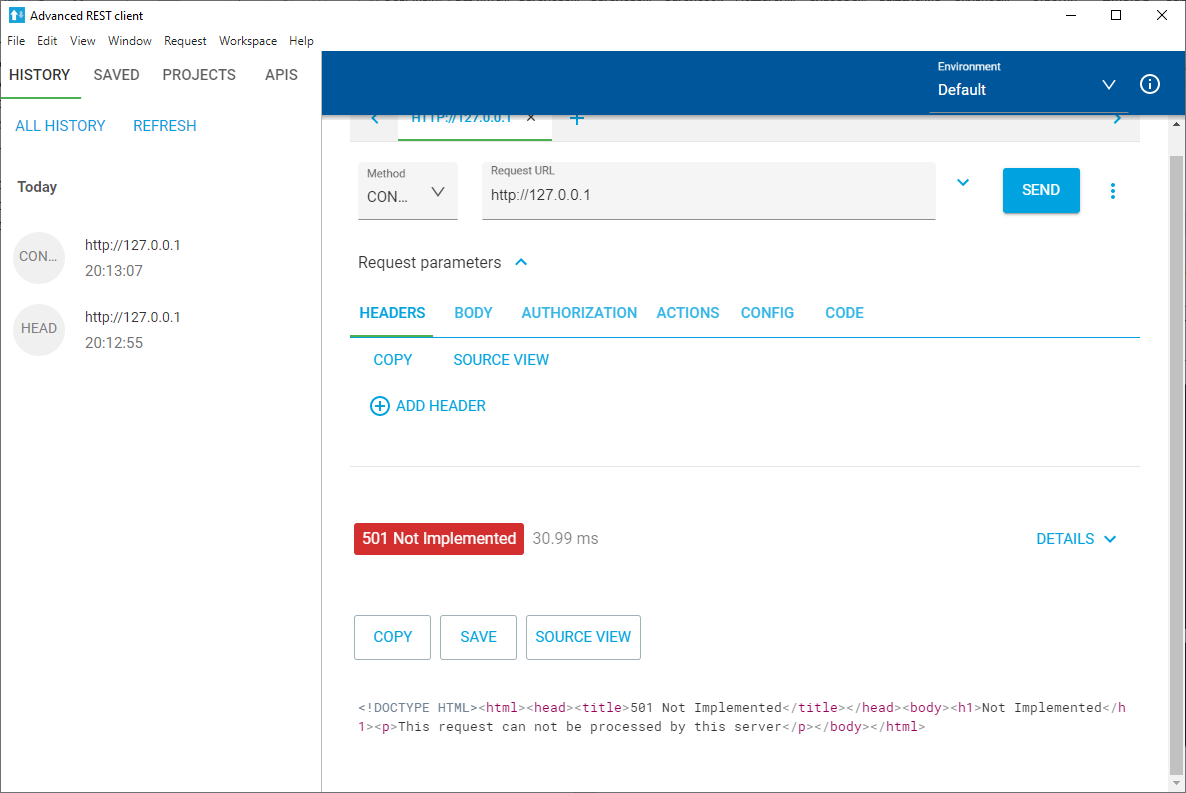
Попробуем зайти по несуществующему адресу:

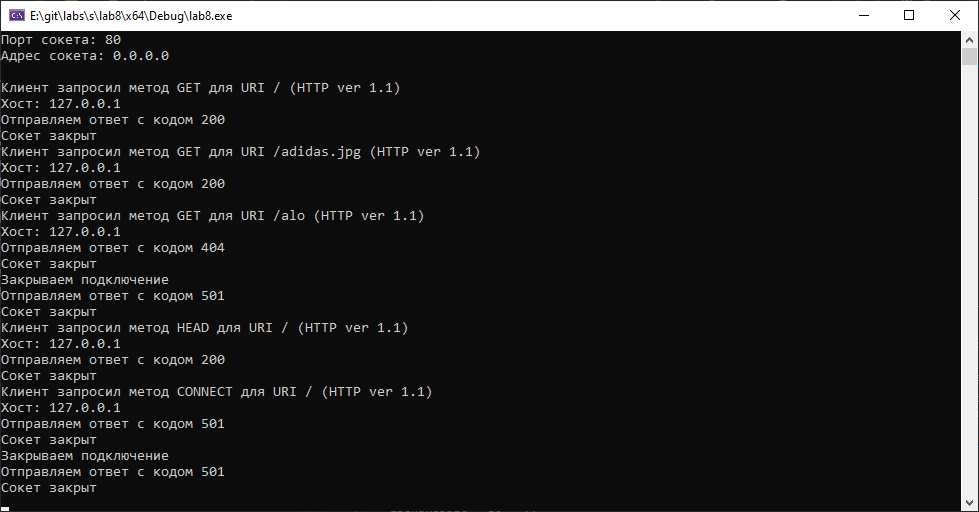


Вывод программы при вышеуказанных запросах:



Поскольку программа реализует только методы GET и HEAD, попробуем отправить запрос с методом CONNECT с помощью ARC:





Как видно из скриншотов выше, произошла ошибка 501.

**Вывод:** в процессе выполнения лабораторной работы был изучен протокол HTTP.

HTTP (Hyper Text Transfer Protocol – протокол передачи гипертекста) – протокол прикладного уровня стека протоколов TCP/IP, предназначенный для передачи данных по сети с использованием транспортного протокола TCP. Протокол HTTP может использоваться также в качестве «транспорта» для других протоколов прикладного уровня. Метод протокола HTTP – это команда, передаваемая HTTP-клиентом HTTP-серверу.

**Контрольные вопросы**

1. Как расшифровывается аббревиатура HTTP?

Hyper Text Transfer Protocol

1. Какой уровень занимает протокол в стеке TCP/IP?

HTTP – это протокол прикладного уровня, использующийся поверх транспортного TCP.

1. На какой технологии построен протокол HTTP?

Данный протокол использует технологию клиент-сервер.

1. Какие преимущества протокола HTTP?

Данный протокол дает возможность гибко задавать способ представления ресурса по формату, языку, кодировке и прочему, что позволяет использовать его как транспорт для других протоколов прикладного уровня.

1. Какие недостатки протокола HTTP?

Среди недостатков можно выделить такие:

Нет возможности точно узнать, какие ресурсы предоставляются сервером.

Для обращения к серверу нужно знать его URI.

1. Какие методы существуют в протоколе HTTP?

GET, HEAD, POST, PUT, OPTIONS, DELETE и прочие. При этом нет необходимости реализовывать их все, а так же возможно использовать другие.

1. Какие нововведения содержит версия HTTP 1.1?

HTTP получил поддержку постоянного соединения, ТСР-сессия остается открытой после ответа, а это позволяет посылать несколько запросов за одно соединение.

1. Какова структура протокола HTTP? Охарактеризуйте каждый элемент.

Состоит их таких элементов:

- Стартовая строка, которая определяет тип сообщения, метод, URI и версию протокола.

- Заголовок, который содержит служебные данные (параметр – значение)

- Тело сообщения, содержащее полезную информацию. Данная часть может отсутствовать.

1. Какие существуют классы кодов состояния?

Существует 5 классов состояния:

1\_\_ - Информационный

2\_\_ - Успешное выполнение операции

3\_\_ - Перенаправление запроса

4\_\_ - Ошибка на стороне клиента

5\_\_ - Ошибка на стороне сервера

1. Какие существуют группы заголовков HTTP?

- General Headers («Основные заголовки») — могут включаться в любое сообщение клиента и сервера;  
- Request Headers («Заголовки запроса») — используются только в запросах клиента;  
- Response Headers («Заголовки ответа») — только для ответов от сервера;  
- Entity Headers («Заголовки сущности») — сопровождают каждую сущность сообщения.

**Листинг программы**

#undef UNICODE

#define WIN32\_LEAN\_AND\_MEAN

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

#include <signal.h>

#include <sstream>

#include <fstream>

#pragma comment (lib, "Ws2\_32.lib")

#pragma comment (lib, "Mswsock.lib")

#define DEFAULT\_BUFLEN 1024

#define DEFAULT\_PORT "http"

DWORD WINAPI ServeClient(LPVOID arg) {

SOCKET ClientSocket = (\*(SOCKET\*)arg);

char request\_buffer[DEFAULT\_BUFLEN];

int request\_buffer\_len = DEFAULT\_BUFLEN;

int iResult;

int iSendResult;

char method[DEFAULT\_BUFLEN];

char URI[DEFAULT\_BUFLEN];

char host[DEFAULT\_BUFLEN];

int ver\_h = 0;

int ver\_l = 0;

iResult = recv(ClientSocket, request\_buffer, request\_buffer\_len, 0);

if (iResult > 0) {

request\_buffer[iResult] = '\0';

sscanf(request\_buffer, "%s %s HTTP/%i.%i\nHost: %s", method, URI, &ver\_h, &ver\_l, host);

printf("Клиент запросил метод %s для URI %s (HTTP ver %i.%i)\nХост: %s\n", method, URI, ver\_h, ver\_l, host);

}

else if (iResult == 0)

printf("Закрываем подключение\n");

else {

printf("Ошибка!");

closesocket(ClientSocket);

WSACleanup();

return 1;

}

char file\_name[DEFAULT\_BUFLEN];

if (strcmp(URI, "/") == 0) {

strcpy(file\_name, "index.html");

}

else {

strcpy(file\_name, URI + 1);

}

std::ifstream file;

file.open(file\_name, std::ifstream::in);

std::stringstream response;

std::stringstream responseBody;

int responseCode = 0;

if (strcmp(method, "GET") && strcmp(method, "HEAD")) {

responseCode = 501;

responseBody

<< "<!DOCTYPE HTML>"

<< "<html>"

<< "<head>"

<< "<title>501 Not Implemented</title>"

<< "</head>"

<< "<body>"

<< "<h1>Not Implemented</h1>"

<< "<p>This request can not be processed by this server</p>"

<< "</body>"

<< "</html>";

response << "HTTP/1.1 501 Not Implemented\r\n"

<< "Version: HTTP/1.1\r\n"

<< "Content-Type: text/html; charset=utf-8\r\n"

<< "Content-Length: " << responseBody.str().length()

<< "\r\n\r\n"

<< responseBody.str();

}

else {

if (file.is\_open()) {

responseCode = 200;

char buffer[DEFAULT\_BUFLEN];

while (file.getline(buffer, DEFAULT\_BUFLEN))

responseBody << buffer;

response << "HTTP/1.1 200 OK\r\n" << "Version: HTTP/1.1\r\n" << "Content-Type: text/html; charset=utf-8\r\n" << "Content-Length: " << responseBody.str().length() << "\r\n\r\n";

if (strcmp(method, "GET") == 0) {

response << responseBody.str();

}

file.close();

}

else {

responseCode = 404;

responseBody << "<!DOCTYPE HTML>" << "<html>" << "<head>" << "<title>404 Not Found</title>" << "</head>" << "<body>" << "<h1>NE MOGU</h1>" << "<p></p>" << "</body>" << "</html>";

response << "HTTP/1.1 404 Not Found\r\n" << "Version: HTTP/1.1\r\n" << "Content-Type: text/html; charset=utf-8\r\n" << "Content-Length: " << responseBody.str().length() << "\r\n\r\n" << responseBody.str();

}

}

printf("Отправляем ответ с кодом %i\n", responseCode);

iSendResult = send(ClientSocket, response.str().c\_str(), response.str().length(), 0);

if (iSendResult == SOCKET\_ERROR) {

printf("Отправка не удалась :с");

closesocket(ClientSocket);

WSACleanup();

return 1;

}

iResult = shutdown(ClientSocket, SD\_SEND);

if (iResult == SOCKET\_ERROR) {

closesocket(ClientSocket);

WSACleanup();

return 1;

}

closesocket(ClientSocket);

printf("Сокет закрыт\n");

free(arg);

}

void intHandler(int dummy) {

exit(0);

}

int main(void){

setlocale(LC\_ALL, "rus");

signal(SIGINT, intHandler);

WSADATA wsaData;

int iResult;

SOCKET ListenSocket = INVALID\_SOCKET;

struct addrinfo\* result = NULL;

struct addrinfo hints;

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

hints.ai\_flags = AI\_PASSIVE;

iResult = getaddrinfo(NULL, DEFAULT\_PORT, &hints, &result);

if (iResult != 0) {

WSACleanup();

return 1;

}

ListenSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

if (ListenSocket == INVALID\_SOCKET) {

freeaddrinfo(result);

WSACleanup();

return 1;

}

printf("Сокет создан успешно!\n");

iResult = bind(ListenSocket, result->ai\_addr, (int)result->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

freeaddrinfo(result);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

printf("Порт сокета: %hu\n",

ntohs(((struct sockaddr\_in\*)(result->ai\_addr))->sin\_port)

);

char tmpbuf[DEFAULT\_BUFLEN];

inet\_ntop(AF\_INET, &(((struct sockaddr\_in\*)(result->ai\_addr))->sin\_addr), tmpbuf, DEFAULT\_BUFLEN);

printf("Адрес сокета: %s\n\n", tmpbuf);

freeaddrinfo(result);

while (1) {

iResult = listen(ListenSocket, SOMAXCONN);

if (iResult == SOCKET\_ERROR) {

closesocket(ListenSocket);

WSACleanup();

return 1;

}

SOCKET\* ClientSocket = new SOCKET;

(\*ClientSocket) = accept(ListenSocket, NULL, NULL);

if (\*ClientSocket == INVALID\_SOCKET) {

closesocket(ListenSocket);

WSACleanup();

return 1;

}

HANDLE thread = CreateThread(NULL, 0, ServeClient, ClientSocket, 0, NULL);

}

closesocket(ListenSocket);

WSACleanup();

return 0;

}