

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе
«Разработка серверного приложения HTTP»
по дисциплине «Сети ЭВМ и телекоммуникации»

Работу выполнил:
студент гр. 43501/3
Хуторной Я. В.

Руководитель:
Вылегжанина К. Д.
«__» _____ 2016 г

Санкт-Петербург
2016

1. Техническое задание

Разработать приложение для операционных систем семейства Windows или Linux, обеспечивающее базовые функции сервера протокола HTTP (Web-сервера).

2. Основные возможности

Приложение должно реализовывать следующие функции:

- Обработка подключения клиента
- Разбор строки URL
- Выдача клиенту запрошенного ресурса
- Обеспечение параллельной загрузки клиенту страниц и медиа-элементов
- Обеспечение параллельной работы нескольких клиентов
- Отображение параметров, передаваемых вместе с методом POST
- Формирование необходимых заголовков протокола HTTP
- Протоколирование соединения клиента с сервером

3. Методика тестирования

В рабочий каталог сервера помещается содержимое какого-либо Web-сайтов сети Internet. Используемый сайт должен иметь несколько уровней вложенности и содержать медиа-элементы (например, графические изображения).

Для тестирования приложения следует использовать стандартные браузеры, имеющиеся в лаборатории (Mozilla Firefox, Internet Explorer, Opera). В процессе тестирования проверяются основные возможности сервера по передаче Web-страниц, медиа-элементов, тестируется корректность разбора строки URL, контролируются параметры, передаваемые с помощью метода POST.

4. Теоретическое обоснование

HTTP - протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных). Основой HTTP является

технология «клиент-сервер», то есть предполагается существование клиентов, которые иницируют соединение и посылают запрос, и серверов, которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) 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.

Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов HTTP использует глобальные URI. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами (например, «куки» на стороне клиента, «сессии» на стороне сервера).

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

- Стартовая строка (англ. Starting line) - определяет тип сообщения;
- Заголовки (англ. Headers) - характеризуют тело сообщения, параметры передачи и прочие сведения;
- Тело сообщения (англ. Message Body) - непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Заголовки и тело сообщения могут отсутствовать, но стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа.

Строка запроса выглядит так:

Метод URI HTTP/Версия

Метод HTTP - последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами.

Код состояния является частью первой строки ответа сервера. Он представляет собой целое число из трёх цифр. Первая цифра указывает на

класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа.

В настоящее время выделено пять классов кодов состояния.

- 1xx Informational («Информационный»)
- 2xx Success («Успех»)
- 3xx Redirection («Перенаправление»)
- 4xx Client Error («Ошибка клиента»)
- 5xx Server Error («Ошибка сервера»)

Заголовки HTTP - это строки в HTTP-сообщении, содержащие разделённую двоеточием пару параметр-значение. Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

Все заголовки разделяются на четыре основных группы:

- 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.

Тело HTTP сообщения если оно присутствует, используется для передачи тела объекта, связанного с запросом или ответом. Тело сообщения отличается от тела объекта только в том случае, когда применяется кодирование передачи, что указывается полем заголовка Transfer-Encoding.

4. Структура приложения

Приложение написано на языке с++. Используется многопоточность для возможности подключения нескольких клиентов. Приложение использует адрес 127.0.0.1 и порт 8000.

5. Тестирование приложения

Для демонстрации приложения была создана HTML-страница с несколькими уровнями вложенности и медиа-элементами.

Первая страница (first.html):

`<html>`

```

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ярослав Хуторной</title>
<style>
/* здесь описаны стили оформления ссылок*/
a:link {
color: #0033FF;
text-decoration: none;
}
a:hover {
color:#666633;
text-decoration: underline;
}
/* здесь описаны стили оформления контента*/
#kontent {
text-align:center;
}
</style>
</head>
<body id="kontent">
<h1 align="center">Хуторной Ярослав</h1>
<p align="center">Offical Page</p>
<p align="center"><a href="http://127.0.0.1:8000/second.html"
title="Медиа">Медиа</p>
<p align="center"><a href="http://vk.com/id40840612" title="Моя страница
вконтакте">Моя страница вконтакте</p>
<p align="center"><a href="http://www.youtube.com/channel/UCzKu_UbDDuf-
ZmUa2Ofg7sQ" title="Мой канал YouTube">Мой канал YouTube</p>
<p </p>
<p </p>
<p </p>
</body>

<body background="photo.jpg" align="center" width="250" height="150">
</html>

```

Вторая страница (second.html):

```

<html>

```

```

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ярослав Хуторной</title>
<style>
/* здесь описаны стили оформления ссылок*/
a:link {
color: #0033FF;
text-decoration: none;
}
a:hover {
color:#666633;
text-decoration: underline;
}
/* здесь описаны стили оформления контента*/
#kontent {
text-align:center;
}
</style>
</head>

<body id="kontent">
<h1 align="center">Хуторной Ярослав</h1>
<p align="center">Offical Page</p>
<p align="center"><a href="http://127.0.0.1:8000/first.html" title="Вернутся на главную">Вернутся на главную</a>
</p>

<body background="photo3.jpg" align="center" width="250" height="150">

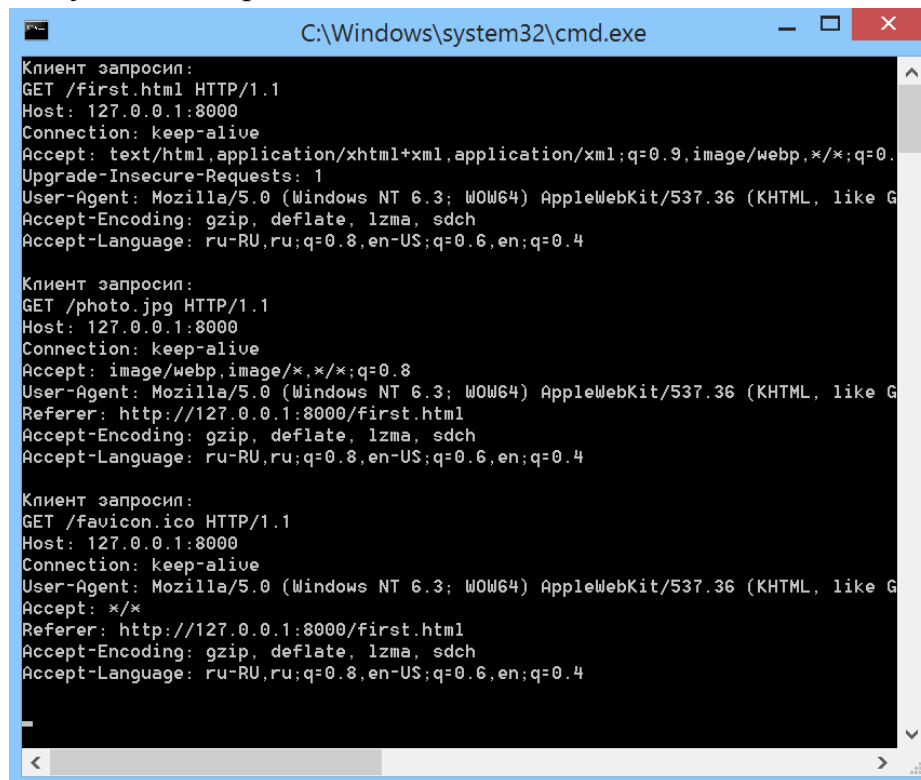
<a href="photo2.jpg" target = "_blank" >Лето2015</a>

<form name="myform" method="post">
<input type="text" name="mytext" size="50">
<input name="Submit" type="submit" value="Отправить сообщение">
</form>
</body>

</html>

```

Запустим приложение и откроем страницу сервера. Для этого введем следующую строку URL в браузере - `http://127.0.0.1:8000/first.html`. На сервер поступят следующие запросы с методом GET:



```
C:\Windows\system32\cmd.exe

Клиент запросил:
GET /first.html HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Accept-Encoding: gzip, deflate, lzma, sdch
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4

Клиент запросил:
GET /photo.jpg HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
Accept: image/webp,image/*/*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Referer: http://127.0.0.1:8000/first.html
Accept-Encoding: gzip, deflate, lzma, sdch
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4

Клиент запросил:
GET /favicon.ico HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Accept: */*
Referer: http://127.0.0.1:8000/first.html
Accept-Encoding: gzip, deflate, lzma, sdch
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4
```

Рис. 1. Метод GET на сервере

Результат ответа сервера на запрос клиента приведен на рис. 2.

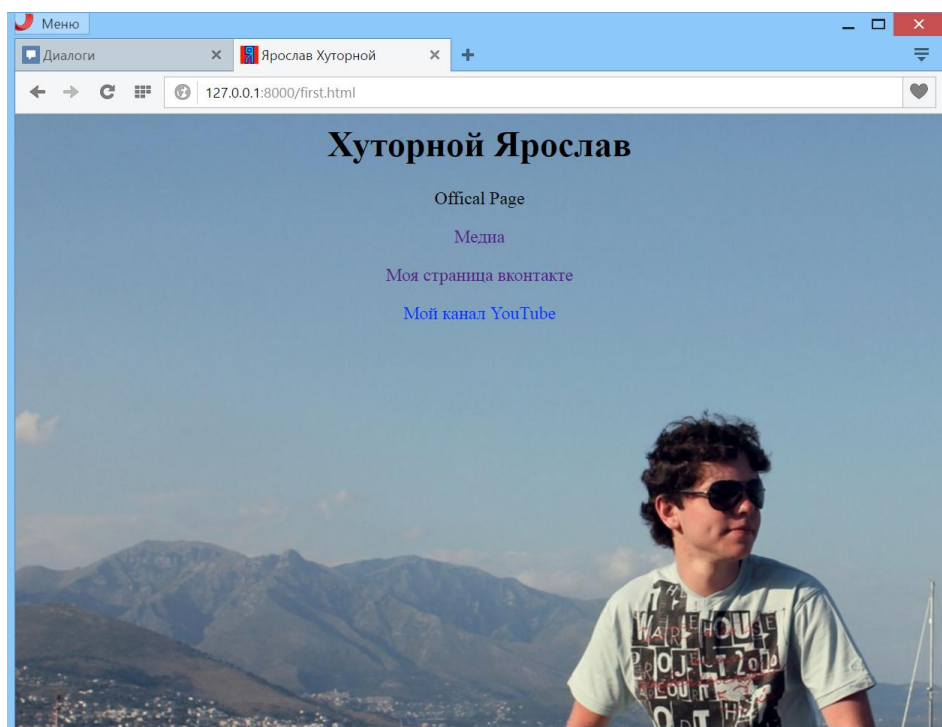
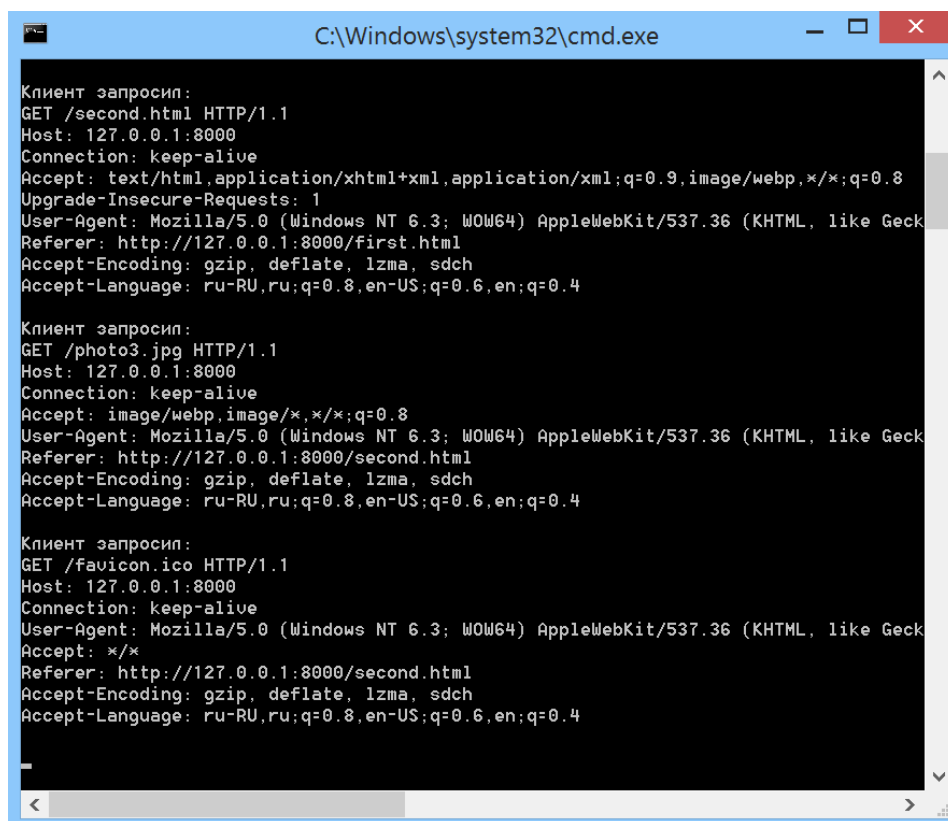


Рис. 2. Результат ответа сервера на запрос клиента

Перейдем по ссылке "Медиа", ссылающуюся на страницу second.html. Результат работы сервера приведены на рис. 3.



```
C:\Windows\system32\cmd.exe

Клиент запросил:
GET /second.html HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Referer: http://127.0.0.1:8000/first.html
Accept-Encoding: gzip, deflate, lzma, sdch
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4

Клиент запросил:
GET /photo3.jpg HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
Accept: image/webp,image/*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Referer: http://127.0.0.1:8000/second.html
Accept-Encoding: gzip, deflate, lzma, sdch
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4

Клиент запросил:
GET /favicon.ico HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Accept: */*
Referer: http://127.0.0.1:8000/second.html
Accept-Encoding: gzip, deflate, lzma, sdch
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4
```

Рис. 3. Метод GET на сервере

Результат ответа сервера на запрос клиента приведен на рис. 4.

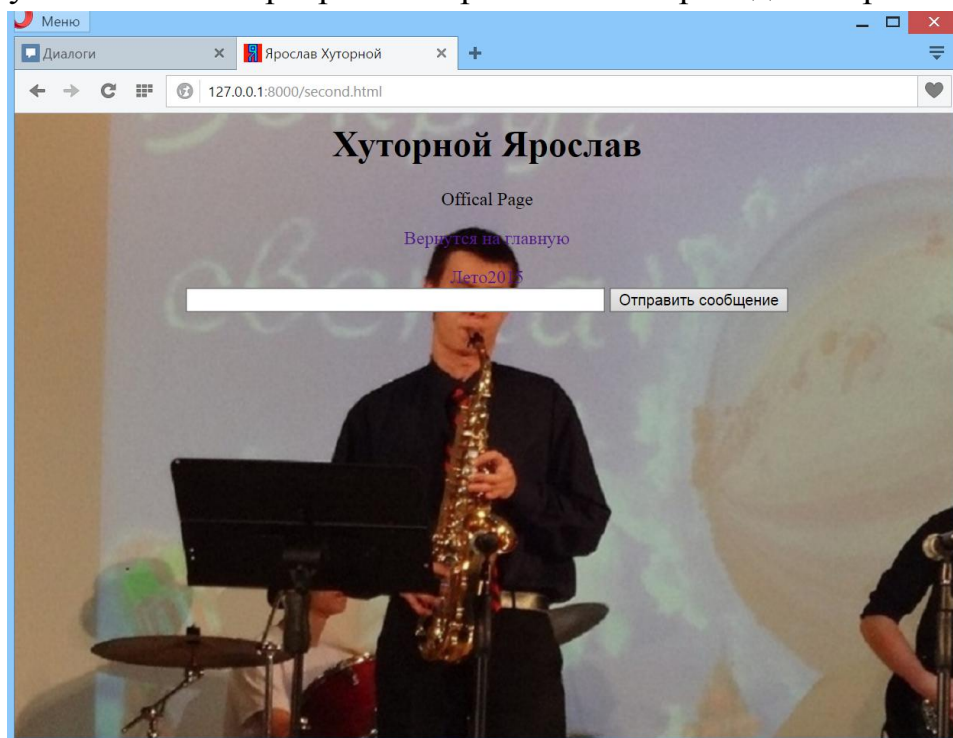


Рис. 4. Результат ответа сервера на запрос клиента

Для демонстрации работы метода POST введем сообщение в строку для ввода и нажмем кнопку "Отправить сообщение".

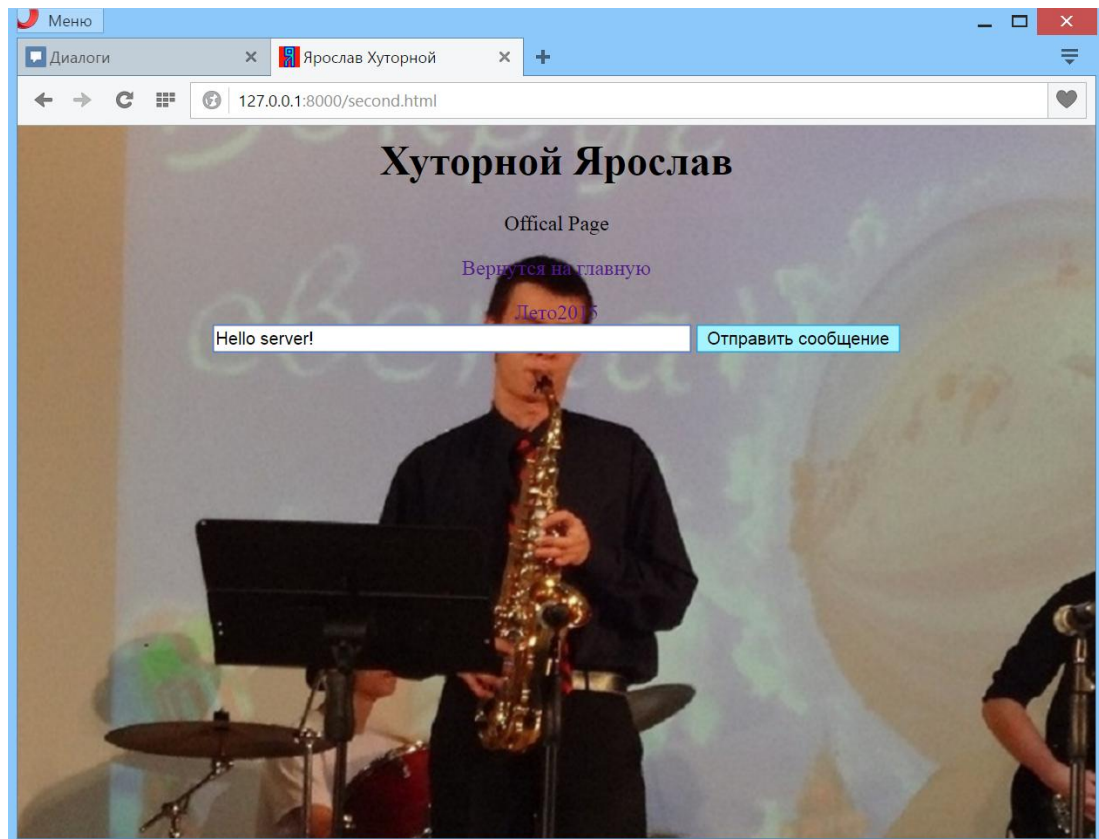


Рис. 5. Посылка запроса со стороны клиента с методом POST

Сервер при этом принимает данные, заключенные в тело сообщения.

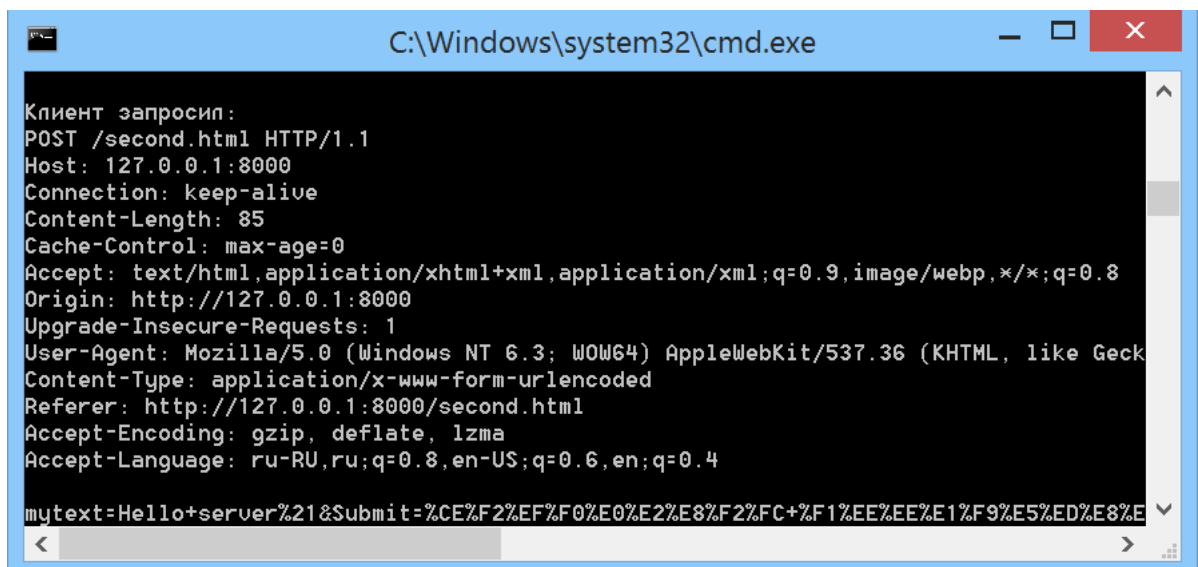


Рис. 5. Результат обработки метода POST на сервере

Попробуем запросить несуществующий файл. Сервер при этом ответит сообщением "FILE NOT FOUND" (рис. 6).

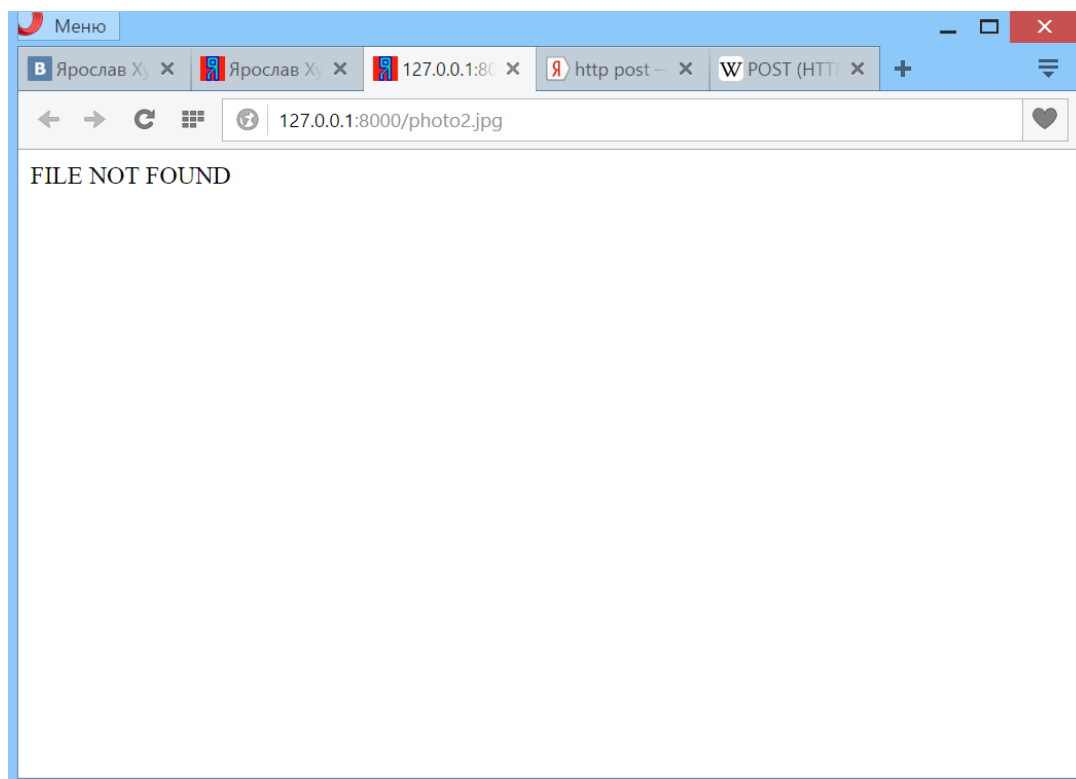


Рис. 6. Результат запроса несуществующего файла

Вывод

Был изучен протокол HTTP и написано приложение HTTP-сервер на языке C++. Тестирование выполнено успешно, ошибок не обнаружено.

Приложение. Листинг программы.

```
#include <iostream>
#include <sstream>
#include <string>
#include <fstream>

#define _WIN32_WINNT 0x501

#include <WinSock2.h>
#include <WS2tcpip.h>

#pragma comment(lib, "Ws2_32.lib")

using std::cerr;
using namespace std;

DWORD WINAPI clientHandler(LPVOID);

int err_notfile(int client_socket)
{
    std::stringstream response;
    std::stringstream response_body;

    response_body<<"FILE NOT FOUND";

    response
    << "HTTP/1.1 500 FILE NOT FOUND\r\n"
    << "Version: HTTP/1.1\r\n"
    << "Content-Type: text/html; charset=windows-1251\r\n"
    << "Content-Length: " << response_body.str().length()
    << "\r\n\r\n"
    << response_body.str();

    int result = send(client_socket, response.str().c_str(), response.str().length(),
0);
    if (result == SOCKET_ERROR)
    {
        cerr << "send failed: " << WSAGetLastError() << "\n";
    }
    closesocket(client_socket);
    return 0;
}

int init_sock()
{
    sockaddr_in addr;
    WSADATA wsaData;

    if (WSAStartup(0x202, &wsaData))
    {
        printf("WSAStart error %d\n", WSAGetLastError());
        return -1;
    }

    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    addr.sin_port = htons(8000);

    int listen_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_socket == INVALID_SOCKET)
    {
```

```

cerr << "Error at socket: " << WSAGetLastError() << "\n";
WSACleanup();
return 1;
}

int result = bind(listen_socket, ( struct sockaddr * )&addr, sizeof(addr));
if (result == SOCKET_ERROR)
{
    cerr << "bind failed with error: " << WSAGetLastError() << "\n";
    closesocket(listen_socket);
    WSACleanup();
    return 1;
}

if (listen(listen_socket, SOMAXCONN) == SOCKET_ERROR) {
    cerr << "listen failed with error: " << WSAGetLastError() << "\n";
    closesocket(listen_socket);
    WSACleanup();
    return 1;
}

return listen_socket;
}

int main()
{
    setlocale(LC_CTYPE, "RUS");
    int listen_socket = init_sock();

    while(1)
    {
        int client_socket = accept(listen_socket, NULL, NULL);
        if (client_socket == INVALID_SOCKET)
        {
            cerr << "accept failed: " << WSAGetLastError() << "\n";
            closesocket(listen_socket);
            WSACleanup();
            return 1;
        }

        HANDLE t2;
        t2 = CreateThread(NULL, 0, clientHandler, (LPVOID)client_socket, 0, NULL);
        if (t2==NULL)
        {
            printf("Error while creating new thread\n");
            return 1;
        }
    }

    closesocket(listen_socket);
    WSACleanup();
}

int get_html(int client_socket, char *file)
{
    std::stringstream response;
    std::stringstream response_body;
    FILE *f;
    long lSize;
    int writed = 0;
    int summ = 0;
    char buff[2048] = {" "};

    f = fopen(file, "rb");

```

```

    if (f == NULL)
    {
        fputs("Ошибка создания файла\n", stderr);
        err_notfile(client_socket);
        return 0;
    }

    fseek(f, 0, SEEK_END);
    lSize = ftell(f);
    fseek(f, 0, SEEK_SET);

    while(summ<lSize)
    {
        writed = fread(buff,1,1,f);
        response_body << buff;
        summ = summ + writed;
    }

    fclose(f);

    response
        << "HTTP/1.1 200 OK\r\n"
        << "Version: HTTP/1.1\r\n"
        << "Content-Type: text/html; charset=windows-1251\r\n"
        << "Content-Length: " << response_body.str().length()
        << "\r\n\r\n"
        << response_body.str();

    int result = send(client_socket, response.str().c_str(), response.str().length(),
0);
    if (result == SOCKET_ERROR)
    {
        cerr << "send failed: " << WSAGetLastError() << "\n";
    }
    closesocket(client_socket);

return 0;
}

int get_file(int client_socket, char *file)
{
    FILE *f;
    long lSize;
    int writed = 0;
    int summ = 0;
    char buff[2048] = {" "};

    f = fopen(file, "rb");
    if (f == NULL)
    {
        fputs("Ошибка создания файла\n", stderr);
        err_notfile(client_socket);
        return 0;
    }

    fseek(f, 0, SEEK_END);
    lSize = ftell(f);
    fseek(f, 0, SEEK_SET);

    while(summ<lSize)
    {
        writed = fread(buff,1,2048,f);
        send(client_socket, buff, 2048, NULL);
        summ = summ + writed;
    }

```

```

    }
    fclose(f);
    closesocket(client_socket);

    return 0;
}

DWORD WINAPI clientHandler(LPVOID args)
{
    std::stringstream response;
    std::stringstream response_body;

    int client_socket = (SOCKET)args;

    const int max_client_buffer_size = 1024;
    char buf[max_client_buffer_size];
    char buf2[max_client_buffer_size];

    int result = recv(client_socket, buf, max_client_buffer_size, 0);
    if (result == SOCKET_ERROR)
    {
        cerr << "recv failed: " << result << "\n";
        closesocket(client_socket);
    }
    else if (result == 0)
    {
        cerr << "connection closed...\n";
    }
    else if (result > 0)
    {
        buf[result] = '\0';

        strcpy(buf2, buf);

        cout << "Клиент запросил: " << endl;
        cout << buf;

        char *q;
        char *q3;

        q= strtok(buf, " ");
        q= strtok(NULL, " ");
        q= strtok(q+1, " ");

        q3= strtok(buf2, " ");
        q3= strtok(NULL, " ");
        q3= strtok(q3+1, " ");

        char *q2;
        q2 = strtok(q, ".");
        q2 = strtok(NULL, " ");

        if(!strcmp( q2, "html" ))
        {
            get_html(client_socket, q3);
        }

        else if(strcmp( q2, "html" ))
        {
            get_file(client_socket, q3);
        }
    }
}

```

```
    closesocket(client_socket);  
    return 0;  
}
```