МГАПИ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПРИБОРОСТРОЕНИЯ И ИНФОРМАТИКИ

Кафедра “Персональные компьютеры и сети”

В.М.Баканов

СЕТЕВЫЕ ТЕХНОЛОГИИ

Учебно-методическое пособие по
выполнению лабораторных работ

Москва 2013

УДК 681.3

Сетевые технологии: Учебно-методическое пособие по выполнению лабораторных работ / В.М.Баканов. —М.: МГУПИ, 2013. —63 с.: ил.

Предлагаемое учебно-методическое пособие предназначено для подготовки студентов III-V курсов различных форм обучения по специальности “Вычислительные машины, комплексы, системы и сети”. Пособие может использоваться студентами для подготовки к выполнению лабораторных и практических работ по курсу “Сетевые технологии”.

В работе рассматривается использование стандартных и разработка оригинальных сетевых приложений (последнее как с явным использованием сокетов, так и с применением инкапсулирующих их возможности компонентов систем быстрой разработки приложений (RAD – *Rapid Application Design*) Delphi и C++Builder фирмы Borland Int. в среде операционной системы Windows). Знание основ программирования на языках Pascal и C/C++ априори предполагается. Описаны конкретные приемы CGI/ISAPI-сетевого программирования, создание клиентских и серверных приложений и др. Создаваемые сетевые приложения работоспособны в локальной и глобальной сети.

Рецензент: доцент, к.т.н. Брейман А.Д.

СОДЕРЖАНИЕ

Стр.

Введение

Программное обеспечение (ПО) компьютерных сетей - сложное в отладке вследствие затруднений при анализе корректности передаваемых и принимаемых данных в виде циркулирующих по сети пакетов; проблемы имеют место даже при использовании стандартных протоколов, методов и процедур обмена данными.

Одним из путей создания надежных сетевых программ является применение проверенных систем разработки, основанных на поддержании технологией объектно-ориентированного программирования (ООП) механизма сетевых сокетов. В сложных случаях необходим тщательный анализ трафика сети с применением, например, снифферов (анализаторов содержимого трафика сети).

Ниже рассматривается практика создания сетевых приложений с использованием систем быстрой разработки приложений (RAD – *Rapid Application Design*) Delphi и C++Builder фирмы *Borland Int*. (сейчас *Embarcadero Tech., Inc.*) в операционной системе (ОС) Windows. Системы отличаются в основном б**а**зовым языком программирования (C/C++ или Object Pascal), элементарные же “кирпичики” (компоненты) обоих систем практически одинаковы. В начале описания каждой работы приводится информация о соответствующих теоретических предпосылках, в конце – вопросы для самопроверки.

1. *Лабораторная работа № 1.* Анализ трафика компьютерной сети с использованием снифферов

Цель работы – приобретение практических знаний и навыков в перехвате и анализе трафика сегмента компьютерной сети.

Теоретическая часть. Снифферы (sniffers, дословный перевод – "вынюхиватели") являются специализированным ПО, предназначенным для анализа потока сообщений (трафика) компьютерной сети передачи информации [1]. Известными системами подобного рода (но глобального уровня) являются ЭШЕЛОН (североамериканский проект, назначением которого является анализ содержимого линий связей Европы) и СОРМ (тотальное протоколирование трафика русскоязычной Сети). Большинство программ и сервисов (ICQ, TelNet, FTP, HTTP, POP3 и т.д.) пересылают пароль и логин пользователя открытым текстом (кодировка и шифровка отсутствует) и сниффер без труда позволяет перехватывать такие сессии.

К простым ПО подобного класса относится, например комплект SpyNet (simik.lgg.ru/spynet312.exe); в штатную поставку Windows Server и др. входит утилита Network Monitor (устанавливается добавлением сервиса Network Monitor Tools & Agent).

Обычно сетевая карта, работающая в сегменте некоммутируемой Ethernet в принципе "прослушивает" весь трафик своего сегмента сети; однако в нормальном (без установки режима PROMISCUOUS MODE) режиме анализируются лишь первые 48 бит заголовка пакета и, если он не соответствует собственному MAC (*Media Access Control*) – адресу карты, последняя перестает воспринимать пакет, считающийся при этом "чужим". Функциональность сниффера достигается переводом сетевой карты в режим PROMISCUOUS MODE, обеспечивающий перехват всех сообщений, циркулирующих в данном сегменте сети безотносительно MAC-адресов (достигается программной установкой соответствующего бита управляющего регистра карты). В случае коммутируемого Ethernet перевод карты в PROMISCUOUS MODE не позволяет прослушивать "чужие" сообщения, в этом случае используется технология "ARP-спуфинга" (путем соответствующей подделки ARP-сообщений данная сетевая карта "притворяется" маршрутизатором с MAC-адресом, однако, данной карты), при этом трафик всех составляющих сегмент сети насильственно направится в сторону карты-обманщика.

Ниже приведен максимально упрощенный исходный текст небольшой программы - сниффера для Unix/Linux (единственный параметр командной строки – имя сетевого интерфейса карты).

```
/* сниффер пишет на stdout все, что захватывает  */
```

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <net/if.h>
#include <unistd.h>
#include <signal.h>
#include <stdio.h>

static volatile int done;

void
handler(int signum)
{
 done = 1;
} /* конец обработчика сигнала HANDLER */

int
main(int argc, char **argv)
{
 char buff[0x10000];
 struct ifreq ifr;
 int s, n;
 if (argc < 2)
  {
    fprintf(stderr, "Usage: %s <interface>\n", argv[0]);
    return 1;
  }
 s = socket(PF_INET, SOCK_PACKET, htons(0x0003));
 if (s == -1)
 {
   perror("socket");
   return 1;
 }
 strcpy(ifr.ifr_name, argv[1]);
 if (ioctl(s, SIOCGIFFLAGS, &ifr) < 0)
 {
   perror("ioctl(SIOCGIFFLAGS)");
   return 1;
 }
/**********************************************************************************************/
 ifr.ifr_flags |= IFF_PROMISC; /* установка режима перехвата */
                               /* ВСЕХ пакетов, поступающих */
                               /* на сетевую карту */
 if (ioctl(s, SIOCSIFFLAGS, &ifr) < 0)
 {
   perror("ioctl(SIOCGIFFLAGS)");
```

```
  return 1;
 }
 signal(SIGINT, handler);
 puts("starting capturing:\n");
 fflush(stdout);
 for (done = 0; !done; )
 {
  n = read(s, buff, sizeof(buff)); /* считываем перехваченный трафик в буфер buff */
  if ( n != -1 )
    write(STDOUT_FILENO, buff, n);
 }
 ifr.ifr_flags &= ~IFF_PROMISC; /* снятие режима перехвата всех пакетов */
 if (ioctl(s, SIOCSIFFLAGS, &ifr) < 0)
 {
  perror("ioctl(SIOCGIFFLAGS)");
  return 1;
 }
 close(s);
 printf("Finished\n");
 return 0;
}  /* конец MAIN */
```

Один из примеров выдачи (определенным образом скомпонованной) информации сниффером приведен ниже; распечатка содержимого перехваченного пакета (датаграммы) состоит из разделенных двоеточием трех колонок: формат пакета-носителя, имя поля, содержимое поля в десятичном и восьмеричном представлении. Этот пакет содержит 14-байтовый заголовок Ethernet, 20-байтовый IP-заголовок, 20-байтовый TCP-заголовок, заголовок HTTP, оканчивающийся двумя подряд CRLF (0D 0A 0D 0A) и далее собственно данные прикладного уровня (WEB-трафик по протоколу HTTP версии 1.1, [2,4]):

```
ETHER: Destination address: 0000BA5EBA11
ETHER: Source address: 00A0C9B05EBD
ETHER: Frame Length: 1514 (0x05EA)
ETHER: Ethernet Type: 0x0800 (IP)
    IP: Version = 4 (0x4)
    IP: Header Length = 20 (0x14)
    IP: Service Type = 0 (0x0)
    IP: Precedence = Routine
    IP:...0.... = Normal Delay
    IP:....0... = Normal Throughput
    IP:.....0.. = Normal Reliability
    IP: Total Length = 1500 (0x5DC)
    IP: Identification = 7652 (0x1DE4)
    IP: Flags Summary = 2 (0x2)
    IP:.......0 = Last fragment in datagram
```

```
IP:.....1 = Cannot fragment datagram
IP: Fragment Offset = 0 (0x0) bytes
IP: Time to Live = 127 (0x7F)
IP: Protocol = TCP — Transmission Control
IP: Checksum = 0xC26D
IP: Source Address = 10.0.0.2
IP: Destination Address = 10.0.1.201
   TCP: Source Port = Hypertext Transfer Protocol
   TCP: Destination Port = 0x0775
   TCP: Sequence Number = 97517760 (0x5D000C0)
   TCP: Acknowledgement Number = 78544373 (0x4AE7DF5)
   TCP: Data Offset = 20 (0x14)
   TCP: Reserved = 0 (0x0000)
   TCP: Flags = 0x10:.A....
   TCP:..0..... = No urgent data
   TCP:...1.... = Acknowledgement field significant
   TCP:....0... = No Push function
   TCP:.....0.. = No Reset
   TCP:......0. = No Synchronize
   TCP:.......0 = No Fin
   TCP: Window = 28793 (0x7079)
   TCP: Checksum = 0x8F27
   TCP: Urgent Pointer = 0 (0x0)
      HTTP: Response (to client using port 1909)
      HTTP: Protocol Version = HTTP/1.1
      HTTP: Status Code = OK
      HTTP: Reason = OK

          ...и так далее...
```

Сниффер может быть установлен на маршрутизаторе (шлюзе, при этом он перехватывает проходящий через интерфейсы этого шлюза трафик) и на конечном узле сети (перехватывается трафик данного сегмента сети).

Обычно сниффер можно настроить на ‘прослушку’ пакетов по заранее определенным протоколам, портам, диапазону IP, направлению передачи и др.; допустимо указывать “горячие” сочетания символов (слова и словосочетания, наличие которых является признаком подготовки определенного действия – например, атаки на конкретный хост с целью его разрушения или взятия под контроль, договоренности о теракте и т.п.).

Области применения снифферов можно разделить на легальные - отладка ПО сетевого класса, обучение, оптимизация сети (обнаружение проблем и “узких мест”), выявление несанкционированных атак на сервера Сети и нелегальные - перехват важной информации (в первую очередь паролей и login’ов пользователей). Стандартным нелегальным приемом использования снифферa является запуск его на целевом сервере (после удачного root-входа) в скрытом режиме (например, маскируясь под named-канал) и несанкциониро-

ванном анализе перехваченной информации. Для предотвращения подобных сценариев используется средства, осуществляющие шифровку трафика (например, протокол SSH).

Анализ вышеприведенной перехваченной информации в реальном случае (анализ многих тысяч пакетов) чрезвычайно затруднен, поэтому разработаны специальные системы для фильтрации и компоновки перехваченного трафика (например, визуализация имеющегося HTTP-трафика WEB-броузером).

Одной из несложных систем подобного рода является комплект SpyNet, включающий два модуля - CaptureNet (собственно перехват трафика, рис.1) и PeepNet (фильтрация пакетов и визуальное представление содержимого, рис.3). CaptureNet определяет MAC- и IP-адреса сетевой карты и по нажатию Start capture переводит ее в выбранный режим (задается набором флажков Hardware Filter); при этом в верхнем правом фрейме окна выводятся параметры каждого захваченного пакета (показываются время, MAC- и IP-источника и адресата, используемый протокол, порт и др.), в правом нижнем фрейме можно просмотреть содержимое выбранного пакета (восьмеричное), при работе CaptureNet записывает отфильтрованные пакеты в файл (по умолчанию CAPTURE.CAP). Заметим, что фактический сброс информации в этот файл происходит в момент превышения объема буфера (задается выбором вариантов меню Capture|Settings) или при остановке захвата пакетов нажатием Stop capture).

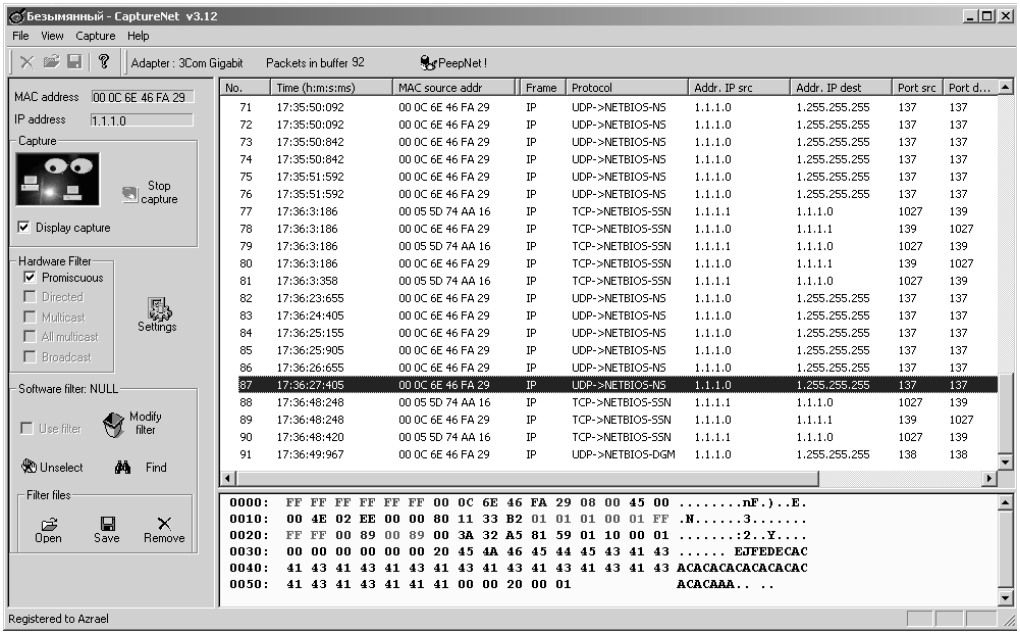

Рисунок 1.— Главное окно утилиты CaptureNet v3.12

В окне Software filter (рис.2, инициализируется кнопкой Modify filter главного окна) задаются параметры фильтрации пакетов: на вкладке Layer 2,3 протокол (например, для перехвата IP/TCP-пакетов целесообразно задать Frame IP,

Layer 3+ TCP), на вкладке Pattern matching – ключевые слова для поиска в пакетах, на вкладке IP addreses – диапазон сканируемых адресов и направление трафика, на вкладке Ports - номера контролируемых портов.

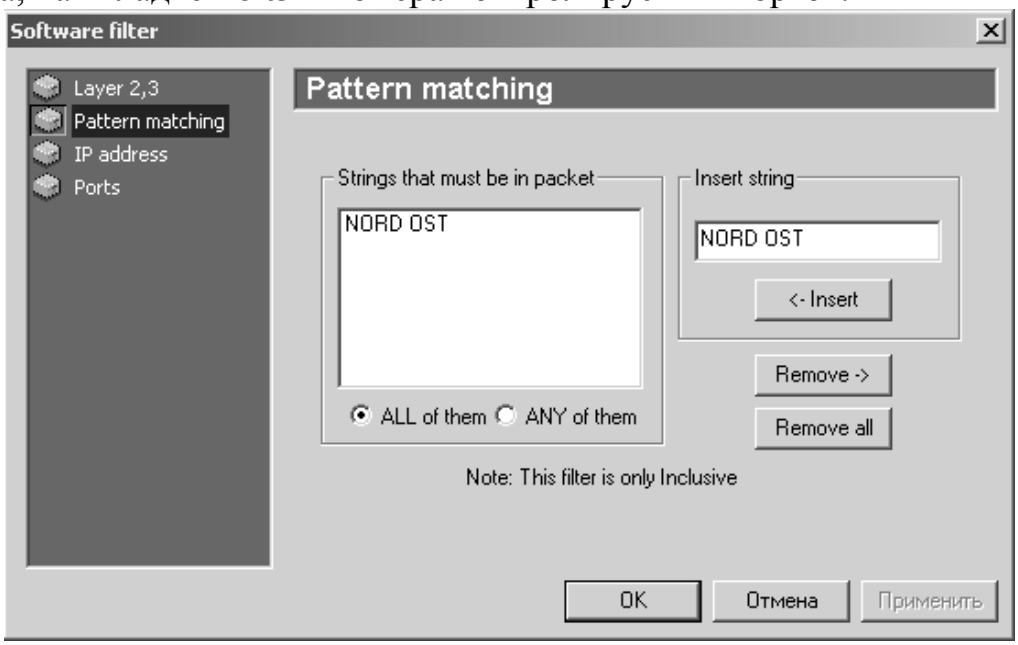

Рисунок 2.— Окно Software Filter утилиты CaptureNet v3.12

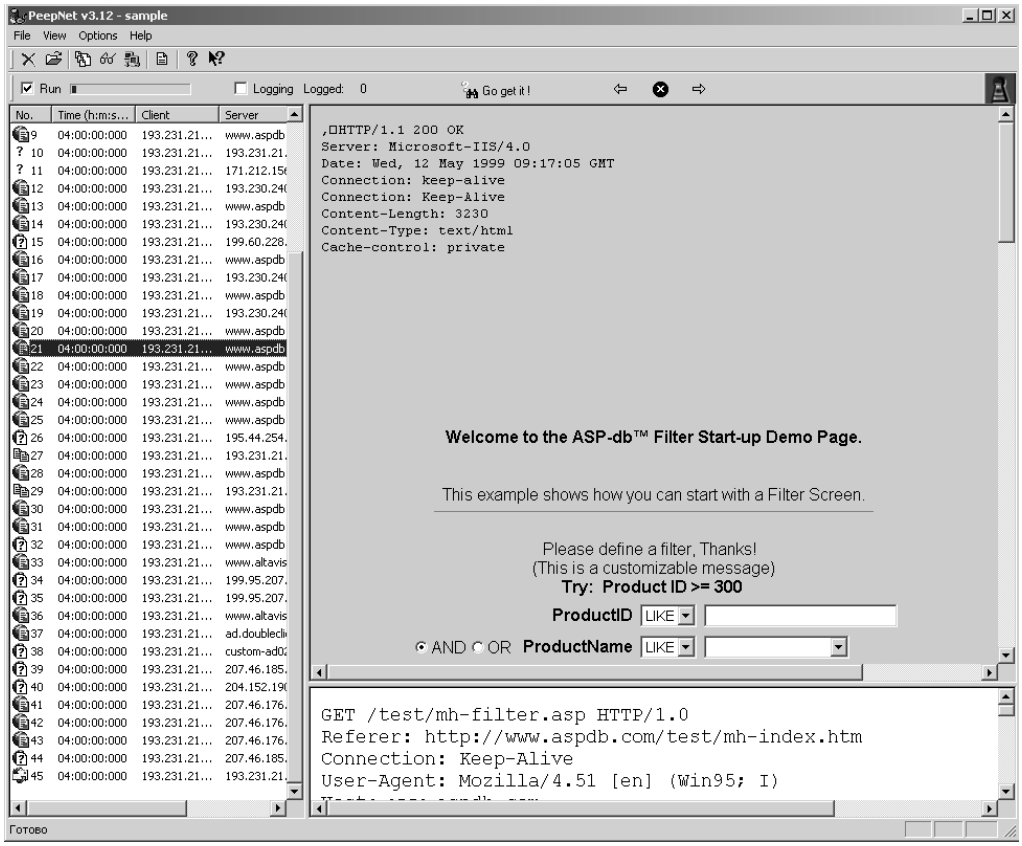

Рисунок 3.— Окно визуализации перехваченного сообщения утилитой PeepNet v3.12

Утилита PeepNet может быть вызвана из CaptureNet или отдельно, при этом может быть проанализирован любой из захваченных CAP-файлов (File|Open), при этом также возможно установить фильтрацию пакетов (Options|Settings). В правом верхнем фрейме выдается графическое представление информационной части пакета в стиле WEB-страницы, в правом нижнем – текст соответствующего запроса клиентской части (рис.3).

Необходимое оборудование – IBM PC-совместимая ЭВМ, предустановленная ОС Windows, пакеты CaptureNet и PeepNet версии 3.12.

Порядок проведения работы. Студент получает задание на анализ трафика локальной сети с заданным условием фильтрации пакетов. Условиями перехвата могут быть, например, только TCP/IP-пакеты, диапазон просматриваемых IP-адрес**о**в, просмотр направленных к конкретному порту пакетов (HTTP – обычно порт 80, FTP – 21, SMTP – 25, SSL – 443, ICQ – 4000 и др.), заданные сочетания символов (USER, PASS, LOG и др.); при этом в сети должны действительно циркулировать указанные пакеты (пакеты ARP- и EtherNet 802.3 обычно присутствуют всегда, осуществляя поддержку сети).

Отфильтрованные пакеты накапливаются в CAP-файле и в дальнейшем анализируются утилитой PeepNet, корректность перехвата и анализа проверяется преподавателем.

Оформление отчета по работе. В отчете указываются заданные преподавателем условия фильтрации пакетов и информация о содержании перехваченных датаграмм. Студент должен сделать выводы о направленности (адресации) пакетов, имеющих широковещательные адреса.

Вопросы для самопроверки.

1. Что представляет из себя ПО класса снифферов и с какой целью применяется?
2. Каковы ограничения методов перехвата информации снифферами?
3. Каким образом сетевая плата конкретной ПЭВМ в локальной сети распознает назначение пакетов по принципу “свой-чужой”?
4. Какие методы применяют с целью исключения возможности перехвата сообщений снифферами?

2. *Лабораторная работа № 2.* Реализация интерактивного режима на HTML-страницах с помощью технологии CGI-расширений WEB-сервера

Цель работы – приобретение практических знаний и навыков в реализации интерактивной работы клиентского броузера и WEB-сервера посредством CGI-технологии.

Теоретическая часть. Классическая технология (последовательность операций) обмена между WEB-сервером и клиентским приложением (броузером) состояла в запросе (с использованием протокола HTTP) броузером нужной HTML-страницы и пересылке запрошенной страницы сервером клиенту. Однако такой режим не позволяет обеспечить (привычный сейчас) интерактивный режим работы пользовательских сайтов. С целью обеспечения интерактивности язык HTML дополнен специальными конструкциями – HTML-формами и вызовами серверных скриптов (*сценариев*), а сам сервер – возможностью расширения функциональности (возможностей).

Основными технологиями расширения возможностей сервера являются CGI (*Common Gateway Interface – Стандартный Шлюзовый Интерфейс*) и ISAPI (*Internet Server Application Program Interface – Программный Интерфейс Приложений Интернет-Сервера*), [2,4]. Программный код согласно обоим технологиям хранится и выполняется сервером, выходная информация генерируется обоими приложениями в виде HTML-предписаний и посылается клиенту. Разница состоит в том, что CGI-приложение оформляется в виде отдельного исполняемого (интерпретируемого при использовании скриптовых языков) файла и выполняется по запросу клиентской части в качестве отдельного процесса (и, соответственно, выполняется в отдельной области ОП); ISAPI-приложение формируется в виде DLL-библиотеки и выполняется в единой с сервером области памяти. С помощью ISAPI реализуются т.н. *фильтры ISAPI*, позволяющие полностью контролировать проходящий через сервер поток данных (с целью шифрования, перекодировки, компрессии данных “на лету”, сбора статистической информации об использовании ресурсов сервера и др.). Т.к. ISAPI-приложения загружаются и выполняются в едином с сервером **а**дресном пространстве, они функционируют в мультизадачном режиме со всеми вытекающими отсюда проблемами (одна из лежащих на поверхности – необходимость использования средств синхронизации при одновременном доступе многих пользователей к единому файлу; чаще применяется механизм *критических секций*).

В табл.1 приведены достоинства и недостатки каждой их рассмотренных технологий, на рис.4 изображена схема потоков данных между клиентским приложением, WEB-сервером и CGI- и ISAPI-приложениями.

Таблица 1. — Сравнение технологий CGI и ISAPI: достоинства и недостатки.

Технология	*Достоинства*	*Недостатки*
CGI	1. Значительно проще программировать и отлаживать. 2. В случае сбоя или аварийного завершения скрипт**а** программа-сервер продолжает успешно функционировать.	При одновременном многократном вызове CGI-скрипт**а** в ОП загружается и выполняется соответствующее число экземпляров программы.
ISAPI	В случае одновременного многократного вызова ISAPI-скрипт**а** инициализируется и выполняется единственный его экземпляр.	1. Программирование и (особенно) отладка затруднены. 2. В случае сб**о**я или аварийного завершения скрипт**а** программа-сервер также аварийно завершается.

Простейшим CGI-приложением может являться, например, DOS-пакетный файл. Например, нижеследующий BAT-файл возвращает (в поток stdout) клиентскому приложению 3 переданных серверу параметра и значения всех определенных в системе переменных среды (посредством исполнения команды set):

```
echo content-type: text/plain
echo
echo %0 &1 %2
set
```

CGI-приложение может быть создано с использование любого удобного языка программирования (часто применяется, например, C/C++). Типичным примером CGI-технологии является использование языка PERL для расширения возможностей сервера, при этом серверная ОС автоматически вызывает PERL-интерпретатор для выполнения PERL-скрипта (известны и средства компиляции исходных PERL-текстов); не менее часто используется система программирования PHP. Принято скрипты помещать в подкаталог с (фиксированным) именем CGI-BIN.

CGI-приложения выводят информацию в стандартный выходной поток stdout (данные из этого потока перехватываются сервером и переправляются клиенту), в методах же приема информации есть разница. В протоколе HTTP определены методы GET и POST передачи данных; в случае GET данные передаются посредством переменной среды с именем QUERY_STRING, для

POST данные передаются через стандартный поток ввода stdout (а длина данных определяется переменной среды с именем CONTENT_LENGTH). В целом метод GET удобен для передачи небольших сообщений (длина переменной среды обычно ограничена), метод POST более предпочтителен вследствие отсутствия ограничений на длину передаваемых сообщений (для передачи данных из локальной ПЭВМ в WWW-сервер используется только метод POST).

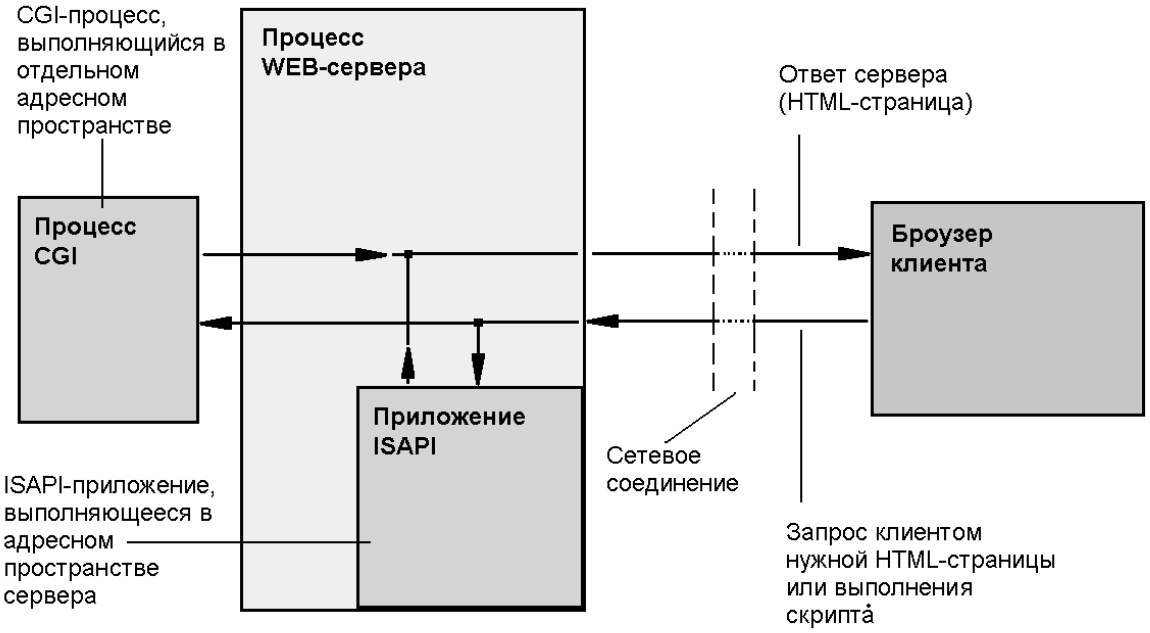

Рисунок 4.— Взаимосвязь между клиентским приложением (броузером), WEB-сервером и его CGI- и ISAPI-расширениями

ISAPI-приложения для принятия приходящих от клиента и выходных данных используют специальные функции собственного API, обеспечивающие б**о**льшую скорость обмена [4,5].

Пример кода HTML-страницы с формой ввода данных представлен ниже:

```
<html>
<body>
<u>Пожалуста, введите Ваши данные:</u>

<FORM ACTION="./cgi_bin/cgi_builder.exe" METHOD="POST">
Имя:
<INPUT TYPE="text" NAME="f_name" VALUE="Peter"><P>
Фамилия:
<INPUT TYPE="text" NAME="l_name" VALUE="Penn"><P>
Возраст:
<INPUT TYPE="text" NAME="old" VALUE="18"><P>
```

```
<INPUT TYPE="submit" VALUE="Послать данные серверу">
</form>

</body>
</html>
```

В данной форме описаны три поля ввода (с именами f_name, l_name, old) и управляющим полем (кнопка, при нажатии на которую данные передаются WWW-серверу); для обработки данных вызывается скрипт ./cgi_bin/cgi_builder.exe, метод обработки – POST).

Переданная HTML-формой информация представлена в виде строки (здесь имяN – имя соответствующего поля формы, значениеN – введенное пользователем значение в соответствующее имяN поле):

```
имя1=значение&имя2=значение2&имя3=значение3
```

В приведенном выше случае эта строка (при нажатии кнопки “Послать данные серверу” без пользовательского ввода) будет иметь вид:

```
f_name=peter&l_name=penn&old=18
```

Заметим, что такая же передаваемая серверу строка соответствует HTML-предписанию (однако метод передачи данных в этом случае будет GET):

```
<a href="./cgi_bin/cgi_builder.exe?f_name=peter&l_name=penn&old=18">Послать данные на сервер</a>
```

Вводимые в поля формы данные могут содержать пробелы (заменяемые в посылаемой строке на ‘+’), символы ‘&’ и запятая кодируются как %26 и %2C соответственно, символы национальных алфавитов представляются в виде шестнадцатиричных кодов вида ‘%XX’. Таким образом, программы CGI и ISAPI обязаны проводить соответствующее декодирование принятой символьной строки.

В случае метода GET значение переданной строки CGI-программа может получить (программа в стиле языка C) посредством операторов:

```
char *sQueryStr;
…..
sQueryStr=getenv(“QUERY_STRING”); // …значение переменной QUERY_STRING
```

а в случае метода POST:

```
int Size;
char sQueryString[4098];
…..
size=atoi(getenv(“CONTENT_LENGTH”)); // длина сообщения
fread(sQueryString, Size, 1, stdin); // скопировать из stdin в буфер sQueryString
```

Заметим, что именно посредством функции getenv(“имя_переменной _окружения”) легко получать значения этих переменных (для последующего анализа); вывод данных в stdout удобно осуществлять посредством операторов printf или fprintf(stdout, …).

Настройка ПЭВМ для проведения работы. При проведении работы используется IBM PC совместимая ЭВМ с установленной поддержкой протокола TCP/IP, сервером Apache и броузером MS Explorer; работа происходит в режиме off-line с иммитацией сети (без выхода во внешнюю сеть).

Удобно работать с виртуальным диском S: (следует включить в режим автозагрузки предписание типа, напр., subst s: d:\internet), при этом рабочий каталог S:\TEST_CGI\, скрипты будут располагаться в S:\TEST_CGI\CGI-BIN\.

IP-адрес локальной ПЭВМ (127.0.0.1) связывается с именем сервера TEST_CGI посредством дополнения файла HOSTS (обычно находится в каталоге C:\WINDOWS\SYSTEM32\DRIVERS\) строкой ‘127.0.0.1 test_cgi’.

WEB-сервер Apache for Windows устанавливается стандартным образом, после установки следует настроить конфигурационный файл HTTP.CONF):

- строку #ServerName new.host.name заменить на ServerName test_cgi
- строку DocumentRoot "C:/Program Files/Apache Group/Apache/htdocs" заменить на DocumentRoot "s:/test_cgi"
- строку ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache/cgi-bin/" заменить на ScriptAlias /cgi-bin/ "s:/test_cgi/cgi-bin/"

Сервер Apache стартуется командной строкой вида

```
"C:\Program Files\Apache Group\Apache\Apache.exe"  -d
                                        C:\PROGRA~1\APACHE~1\Apache –s
```

и закрывается

```
"C:\Program Files\Apache Group\Apache\Apache.exe"  -d
                                    C:\PROGRA~1\APACHE~1\Apache -k shutdown
```

причем “иконки” с соответствующими ссылками создаются автоматически при инсталляции Apache. Удобно использовать комплект программ Denwer, включающий компактный вариант сервера Apache с поддержкой Perl, PHP, MySQL и комплектом документации (http://web.dklab.ru).

Формально проверить связь IP-адреса с виртуальным хостом TEST_CGI можно путем тестирования системы командой PING TEST_CGI.

Необходимое оборудование – IBM PC-совместимая ЭВМ, предустановленная ОС Windows, настроенный (см. выше) пакет Apache for Window, стандартный броузер MS Explorer.

Порядок проведения работы. Студент проверяет корректность установки и настройки сети и сервера Apache, разрабатывает управляющую HTML-страницу с интерактивной формой и скрипты, проверяет функционирование полученной системы.

Часть 1 работы. Первой задачей является проверка функционирования скрипта в виде BAT-файла (см. выше); этот файл (с именем CGI_BAT.BAT) должен находиться в каталоге S:\TEST_CGI\CGI_BIN\. Управляющий (содержащий HTML-форму) файл INDEX.HTML (см. выше) должен находиться в каталоге S:\TEST_CGI\; с целью проверки функционирования CGI в виде пакетного файла INDEX.HTML дополняется строкой:

```
<a href="./cgi-bin/cgi_bat.bat?field1=value1&field2=value2&field3=value3">Вызов CGI BAT-файла с параметрами</a>
```

В командной строке броузера записывается TEST_CGI (т.е. задается присоединение к серверу заданного имени, по умолчанию Apache’а стартовой страницей является INDEX.HTML). При корректной настройке выдача сервера (на экран броузера или в отдельный файл на клиентской машине) будет иметь приблизительно следующий вид:

```
s:\test_cgi\cgi-bin>echo Content-type: text/plain
Content-type: text/plain

s:\test_cgi\cgi-bin>echo
Режим вывода команд на экран (ECHO) включен.

s:\test_cgi\cgi-bin>echo "s:/test_cgi/cgi-bin/cgi_bat.bat"
"s:/test_cgi/cgi-bin/cgi_bat.bat"
```

```
s:\test_cgi\cgi-bin>set
COMSPEC=C:\WINDOWS\system32\cmd.exe
DOCUMENT_ROOT=s:/test_cgi/
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_ACCEPT_LANGUAGE=ru
HTTP_CONNECTION=Keep-Alive
HTTP_HOST=test_cgi
HTTP_REFERER=http://test_cgi/index.html
HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
PATH=e:\COSMOS;D:\C_Build6\Bin;D:\C_Build6\Projects\Bpl;C:\WINDOWS\
sytem32;C:\WINDOWS;C:\WINDOWS\system32\WBEM;d:\COSMOS;C:\progra~1\
Perl\bin;d:\C_Build4\CBUILD~1\Projects\Bpl;d:\C_Build4\CBUILD~1\Bin;C:\Program
Files\ATI Technologies\ATI ControPanel;d:\devstudio\sharedide\bin\ide;
d:\devstudio\sharedide\bin;d:\devstudio\vc\bin;d:\dephi43\bin;
c:\progra~1\borland\vbroker\bin;c:\progra~1\borland\vbroker\jre\bin;
c:\;c:\DOS;c:\NC4;e:\tc;e:\DELPHI\BIN;
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.JS;.WS
PROMPT=$P$G
REMOTE_ADDR=127.0.0.1
REMOTE_PORT=1038
SCRIPT_FILENAME=s:/test_cgi/cgi-bin/cgi_bat.bat
SERVER_ADMIN=you@your.address
SERVER_NAME=test_cgi
SERVER_PORT=80
SERVER_SIGNATURE=<ADDRESS>Apache/1.3.6 Server at test_cgi Port
80</ADDRESS>

SERVER_SOFTWARE=Apache/1.3.6 (Win32)
SystemRoot=C:\WINDOWS
WINDIR=C:\WINDOWS
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=field1=value1&field2=value2&field3=value3
REQUEST_URI=/cgi-bin/cgi_bat.bat?field1=value1&field2=value2&field3=value3
SCRIPT_NAME=/cgi-bin/cgi_bat.bat
```

Следует обратить внимание на обилие информации о серверной машине и собственно сети, полученной столь примитивным образом. Выводятся значения всех переменных окружения (заглавные буквы), структура каталогов, протоколов и портов сервера, поддерживаемых кодировок и многое др.

Часть 2 работы. Задачей является создание приложения CGI_CPP.EXE в виде исполняемого файла с помощью системы C++Builder и проверке его функциональности (CGI-программа осуществляет первичный анализ переданной HTML-формой данный и генерирует HTML-файл для возврата клиенту); исполняемый файл CGI_CPP.EXE должен находиться в каталоге S:\TEST_CGI\CGI_BIN\. С целью компиляции в C++Builder ver. 3 или 4 не требующего дополнительных BPL- и DLL-файлов приложения следует в вызываемом посредством выбора в главном меню варианта Project|Options… окне Project Options интегрированной среды снять флаги с указателя Build with runtime packages (закладка Packages) и с указателя Use dynamic RTL (закладка Linker). Исходный текст С-приложения приведен ниже:

```
void DecodeStr(char *sString);
char DecodeHex(char *str);

//---------------------------------------------------------------------------
void
__fastcall TForm1::MainFormCreate(TObject *Sender)
// выполняется при создании формы MainForm
{ FILE *f;
  char sBuff[8196],  // буфер для принятых от броузера данных
       sSubStr[512];  // буфер для каждой подстроки
  int lData; // длина буфера полученных данных
//
//f=fopen("s:\\a.htm", "w+"); // вывод в дисковый файл для отладки приложения
  f=stdout;
//
// вывод строки заголовка HTTP
  fprintf(f, "Content-type: text/html\n\n"); // обязательно \n\n !!!
//
// определение метода передачи параметров (берется из переменной
// среды REQUEST_METHOD)
  char *sMethod=getenv("REQUEST_METHOD");
//
  if (!strcmp(sMethod, "POST")) // это метод POST !!!
   {
    fprintf(f, "Используется метод посылки данных POST<P>");
//
// определение общей длины полученных от броузера данных
// (берется из CONTENT_LENGTH)
    lData=atoi(getenv("CONTENT_LENGTH"));
// копируем эти данные (длиной lData) из входного потока stdin в буфер sBuff
    fread(sBuff, lData, 1, stdin);
    sBuff[lData]='\0'; // закрываем строку нулем
   } // конец взятия данных методом POST
//
  else
```

```
  if (!strcmp(sMethod, "GET")) // это метод GET !!!
   {
    fprintf(f, "Используется метод посылки данных GET<P>");
// получить данные из переменной среды QUERY_STRING
    strcpy(sBuff, getenv("QUERY_STRING"));
    lData=strlen(sBuff);
    sBuff[lData]='\0'; // закрываем строку нулем
   } // конец взятия данных методом GET
//
  else // неизвестный метод...
   {
    fprintf(f, "Метод посылки данных НЕИЗВЕСТЕН”);
    exit(-13);
   }
//
// выводим принятую от броузера строку на динамически
// создаваемую HTML-страницу
  fprintf(f, "<HTML><BODY>\n\n<u>Полученная от броузера строка (до перекоди-
                                          ровки):</u> %s<BR><BR>", sBuff);
//
//  fprintf(f, "<HTML><BODY>\n\n<u>Полученная от броузера строка (после переко-
                                          дировки):</u> %s<BR><BR>", sBuff);
//
  fprintf(f, "<u>Разбираем строку по полям формы (разделитель - символ
                    '&'):</u><P>");
//
// для унификации поиска полей формы дополняем sBuff справа символом '&'
  sBuff[lData]='&';
  sBuff[lData+1]='\0';
//
// в sBuff теперь находятся сочетания типа 'имя поля формы=значение поля&'
//
  int iLeft=0; // номер начального символа подстроки в строке sBuff
//
  for(int i=0; i<strlen(sBuff); i++) // цикл по строке sBuff
   if(sBuff[i]=='&') // нашли очередной разделитель '&'
    {
     for(int k=0,j=iLeft; j<i; j++) // копируем очередную подстроку в sSubStr
       sSubStr[k++]=sBuff[j];
     sSubStr[i-iLeft]='\0'; // закрываем найденную подстроку нулем
     DecodeStr(sSubStr); // раскодирование строки из URL-кода
     iLeft=i+1; // устанавливаем начало следующей подстроки
//
     fprintf(f, "<i>имя поля формы=значение поля:</i> %s<BR>", sSubStr);
    } // конец if (sBuff[]...
//
  fprintf(f, "<BR><BR><HR>Конец динамически сгенерированной HTML-
                                      страницы...");
  fprintf(f, "\n\n</BODY></HTML>");
```

```
  fclose(f);
//
} // конец функции MainFormCreate
//---------------------------------------------------------------------------

// Функция DecodeStr - раскодирование строки из кодировки URL
//
void DecodeStr(char *sString)
{ int src, dst;
  char ch;
  for(src=0,dst=0; sString[src]; src++,dst++) // цикл по строке
   {
    ch = sString[src]; // получаем очередной символ перекодируемой строки
    ch = (ch == '+') ? ' ' : ch; // заменяем символ '+' на пробел
    sString[dst] = ch; // сохраняем результат
// обработка шестнадцатеричных кодов вида %xx
    if(ch == '%')
     {
      sString[dst] = DecodeHex(&sString[src + 1]); // преобразуем %00 в код символа
      src += 2;
     } // конец IF ch…
   } // конец FOR src…
  sString[dst] = '\0'; // закрываем строку нулем
} // конец функции DecodeStr
//---------------------------------------------------------------------------

// функция DecodeHex - раскодирование строки %xx
//
char DecodeHex(char *str)
{ char ch;
// обрабатываем старший разряд
  if(str[0] >= 'A') // здесь и далее 'A' - латиница !!!
   ch = ((str[0] & 0xdf) - 'A') + 10;
  else
   ch = str[0] - '0';
  ch <<= 4; // сдвигаем его влево на 4 бита
  if(str[1] >= 'A') // обрабатываем младший разряд и складываем его со старшим
   ch += ((str[1] & 0xdf) - 'A') + 10;
  else
   ch += str[1] - '0';
  return (ch); // возвращаем результат перекодировки
} // конец функции DecodeHex
//---------------------------------------------------------------------------
```

Компонентная C-функция MainFormCreate создана с помощью Object Inspector'а как вызываемая при событии MainFormCreate (создание единственной в приложении формы MainForm), при этом в управляющем файле CGI_CPP.CPP строка Application->Run(); закомментирована (CGI-приложение

не требует окна). Функции DecodeStr и DecodeHex служат для (вышеописанного) декодирования символов в получаемой от броузера строке.

Скрипт CGI_CPP.EXE определяет метод пересылки данных (GET или POST), выделяет подстроки вида “имя=значение&” и убирает конечный “&”, осуществляет декодирование символов и выдает полученные данные в показанном на рис.5 виде. Дальнейший анализ полученной от клиента информации (не конкретизирован) заключается в анализе пар подстрок “имя=значение” с целью использования введенных пользователем значений (например, для поиска в базе данных и др.).

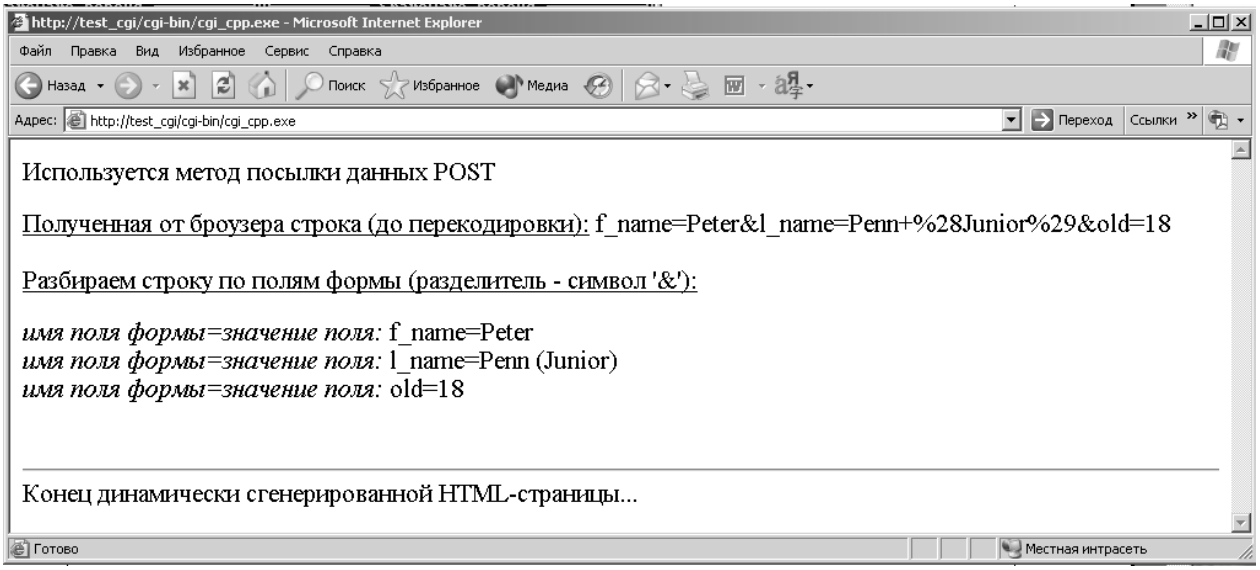

Рисунок 5.— Интерпретация возвращаемой скрипт**о**м CGI_CPP.EXE страницы HTML

С целью более полного уяснения возможностей CGI-скриптов в виде выполняемых модулей возможна модификация исходного С-текста; преподаватель может дать задание на доработку исходного текста, расширяющего функциональность скрипта.

Оформление отчета по работе. В отчете приводится текст выдачи сервера (на экран броузера или в отдельный файл на клиентской машине) информации о сервере и состоянии сети. Исходный текст осуществляющей первичный анализ переданных HTML-формой данных и генерирующей HTML-файл для возврата клиенту программы приводится в случае целенаправленной (с целью расширения функциональности) его модификации.

Вопросы для самопроверки.

1. Назвать основные технологии реализации интерактивности на HTML-страницах.
2. В чем заключается разница между технологиями CGI и ISAPI? Провести

сравнение с целью выявления преимуществ и недостатков CGI и ISAPI.

3. Каковы основные достоинства и недостатки методов POST и GET?
4. Каким образом клиентская сторона передает исполняемому на стороне WEB-сервера скрипту необходимую информацию?
5. Перечислить системы программирования, используемые при разработке исполняемых на стороне сервера скрипт**о**в.

3. *Лабораторная работа № 3.* Создание счетчика посещений сайта на основе CGI-технологии

Цель работы – приобретение практических знаний и навыков в создании практически полезных CGI-скриптов.

Теоретическая часть. Счетчик числа посещений является удобной принадлежностью каждого сайта; и хотя в настоящее время имеется немало методов его реализации, CGI- (и ISAPI-) технологии позволяют наиболее выпукло показать свои возможности.

Общая последовательность реализации счетчика посещений заключается в следующем - на нужной (обычно стартовой) HTML-странице располагается вызов определенного CGI/ISAPI-скрипта, который при каждом открытии этой страницы производит следующие действия:

1. Читает файл (в нижеследующем примере CNTDAT.DAT), содержащий (в виде строки текста) значение числа предыдущих посещений.
2. Увеличивает это значение на 1 и переписывает файл счетчика с новым значением числа посещений.
3. Считывает файл-шаблон (в нашем примере HOME.TM) данной страницы (содержащий ее текст без инструкций счетчика), при этом в нужном месте HTML-текста помещен специальный (уникальный) м**а**ркер (*сигнат**у**ра*, в нашем случае это 5 тильд).
4. Просматривая шаблон, скрипт находит маркер и заменяет его на строковое значение счетчика.
5. Измененная таким образом HTML-страница отправляется на stdout (т.е. клиентскому приложению - броузеру).

Ниже приведен текст программы, осуществляющей эти действия. Как и ранее, программа выполняется в момент создания главной (и единственной) формы приложения, причем при этом в управляющем файле COUNTER.CPP строка Application->Run(); закомментирована:

```
void
__fastcall TForm1::FormCreate(TObject *Sender)
// вызывается при событии OnCreate (создание главной формы TForm1)
{
// Программа COUNTER.C - счетчик посещений сайта
// (C) Фролов А.В., 1997
// Модернизация для C++Builder - Баканов В.М., 2004
// e881e@yahoo.com, http://pilger.mgapi.edu
//
 FILE *TemplateFile; // идентификатор файла шаблона главной страницы
```

```
 FILE *CounterFile; // идентификатор файла счетчика
 DWORD dwFileSize; // размер файла шаблона
 LPSTR szTemplate; // адрес шаблона главной страницы
 CHAR szBuf[10]; // временный буфер для работы со счетчиком
 INT nCounter; // текущее значение счетчика
 LPSTR szCounterPtr; // указатель на поле счетчика в шаблоне
 char UnikMarker[]="~~~~~"; // маркер для замены на номер посещения (5 тильд)
//
// Открываем файл счетчика для чтения
 CounterFile = fopen("CNTDAT.DAT", "r");
// Читаем из файла строку значения счетчика
 fread(szBuf, 7, 1, CounterFile);
// Закрываем файл счетчика
 fclose(CounterFile);
// Преобразуем содержимое счетчика из строки в число
 sscanf(szBuf, "%d", &nCounter);
// Увеличиваем значение счетчика на 1
 nCounter++;
// Записываем в буфер szBuf 5 цифр нового значения счетчика
 sprintf(szBuf, "%05.5ld", nCounter);
// Открываем файл счетчика для записи
 CounterFile = fopen("CNTDAT.DAT", "w");
// Сохраняем новое значение счетчика в файле
 fprintf(CounterFile, "%d", nCounter);
 fclose (CounterFile);
//
// Считываем файл шаблона в оперативную память
//
// Открываем файл шаблона для чтения
 TemplateFile = fopen("HOME.TM", "r");
// Определяем размер файла в байтах
 fseek(TemplateFile, 0, 2); // на конец файла...
 dwFileSize = ftell(TemplateFile); // длина Оного
// Устанавливаем указатель текущей позиции на начало файла шаблона
 fseek(TemplateFile, 0, 0);
// Захватываем память для буфера шаблона
 szTemplate = (char *) malloc(dwFileSize);
// Загружаем шаблон в буфер
 fread(szTemplate, dwFileSize, 1, TemplateFile);
// Заменяем 5 цифр значения счетчика на новые в буфере шаблона
//
// Ищем уникальный маркер поля счетчика UnikMarker
 szCounterPtr = strstr(szTemplate, UnikMarker);
// Копируем в эТО поле новое значение счетчика
 if(szCounterPtr != NULL)
   strncpy(szCounterPtr, szBuf, 5);
// Выводим заголовок HTTP и разделительную строку
 printf ("Content-type: text/html\n\n");
// Выводим шаблон с измененным значением поля счетчика
```

```
 fwrite (szTemplate, dwFileSize, 1, stdout);
// Освобождаем буфер шаблона          '
 free(szTemplate);
} // конец FormCreate
//---------------------------------------------------------------------------
```

Текст шаблона главной страницы (файл HOME.TM) приведен ниже (в качестве уникального маркера используются 5 тильд):

```
<html>
<body>
<h2>Главная страница корпорации MGAPI Int.</h2><p>
Добро пожаловать на нашу главную страницу !
<p>
<hr>
<p>
Вы посетитель номер <font color=red><B>~~~~~~</B></font> с 01 января 1701 года
</body>
<html>
```

Назначение стартового файла состоит всего лишь в передаче управления скрипту, модифицирующего шаблон HOME.TM и выдающего его клиенту в качестве главной страницы. В тексте стартового файла DEFAULT.HTM использованы два (равноценных) варианта вызова скрипта – с использованием тега <META…> или команд JavaScript:

```
<html>
<head>
<META HTTP-EQUIV="REFRESH" CONTENT="1; URL=./cgi-bin/counter.exe?">
</head>
<body>
Это всего лишь пустая страница, служащая для вызова скрипта,
осуществляющего подсчет числа посещений<p>
<script language="JavaScript">
<!--
// выполняется при загрузке страницы
   window.location.href = "./cgi-bin/counter.exe?";
// -->
</script>
</body>
<html>
```

Заметим, что в конце строки обращения к скрипту помещен знак “?”, что намекает на возможность передаче скрипту некоторых значений (например, идентификатора страницы, в этом случае возможно, например, вести счетчик посещений для каждой страницы).

Необходимое оборудование – IBM PC-совместимая ЭВМ, предустановленная ОС Windows, настроенный (см. выше) пакет Apache for Window, стандартный броузер MS Explorer.

Порядок проведения работы. Студент проверяет корректность установки и настройки сети и сервера Apache, создает HTML-страницы и скрипты, проверяет функционирование полученной системы.

Файл скрипта COUNTER.EXE находится в каталоге ./TEST_CGI/CGI-BIN, там же должны располагаться файлы CNTDAT.DAT и HOME.TM; стартовый файл DEFAULT.HTM обычно располагается в каталоге ./TEST_CGI/. Сервер Apache настраивается согласно приведенным в предыдущей работе рекомендациям, работа начинается набором в командной строке броузера строки http://test_cgi/default.htm. Пример интерпретации броузером страницы сайта со счетчиком числа посещений приведен на рис.6 (содержимое командной строки указывает, что страница создана динамически скриптос COUNTER.EXE).

Заметим, что в данном случае файл CNTDAT.DAT является *разделяемым ресурсом* для нескольких экземпляров CGI-программы COUNTER.EXE и при интенсивных запросах (т.н. DDOS – атаки на сервер) работа счетчика посещений может быть нарушена (для устранения этого используются специальные технологии, обеспечивающие доступ к разделяемому ресурсу по записи только одному экземпляру программы в каждый момент). Модернизация программы COUNTER в соответствие с этими требованиями (обычно используется механизм семафоров) является прекрасной самостоятельной работой для студентов.

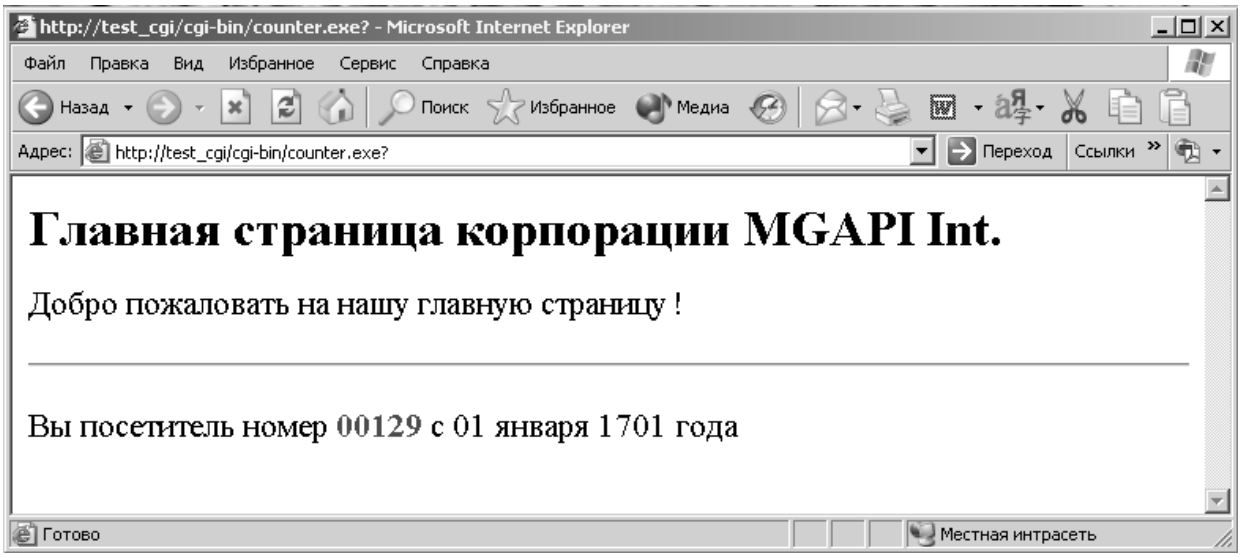

Рисунок 6.— Основанная на шаблоне HOME.TM страница со счетчиком числа посещений

Дополнительными (домашними) заданиями также могут являться:

- Модификация скрипта COUNTER, позволяющая вести счетчик числа посещений для каждой страницы сайта.
- Модификация скрипта, позволяющая организоватьь индивидуальный счетчик посещений для каждого посетителя сайта.
- Разработка скрипта, выдающего число посещений в графическом виде (при этом полезно ознакомиться с работами [3,4]).

Оформление отчета по работе. В отчете приводится последовательность работы системы учета числа посещений главной страницы сайта (с описанием необходимости каждого этапа). Исходный текст выполняющей соответствующие действия С-программы целесообразно привести в случае ее целенаправленной модификации.

Вопросы для самопроверки.

1. Какие пути целенаправленной модификации HTML-страниц принципиально возможны?
2. Предложить программную реализацию метода обеспечения доступа к содержащему текущее значение числа посещений страницы файлу только одному экземпляру модифицирующей программы в каждый момент (по записи)?
3. Каким путем возможно реализовать счетчик посещений для каждой страницы отдельно?

4. *Лабораторная работа № 4.* Разработка клиентского сетевого приложения на основе сокетов (подсоединение к службе даты-времени)

Цель работы – приобретение практических знаний и навыков в разработке сетевых программ, использующих технологию сокетов (*сетевых гнезд*).

Теоретическая часть. Впервые понятие сокета как реализации функций интерфейса прикладного программирования было предложено в университете Беркли, Калифорния (*University of California at Berkeley Sockets API*) при разработке спецификации Berkeley UNIX [4]. Сокет ("гнездо" в дословном переводе) обеспечивает конечную точку соединения и функционирует как двунаправленный канал для входящих и исходящих данных между компьютерами в сети.

При формальном программировании сокет более всего похож на идентификатор файла (*file handle*), который нужен для выполнения над файлом операций чтения или записи (подробнее о программировании сокетов см. [2,5]). Для ОС Windows разработан набор функций Windows Socket (*WinSock API*), формально оформленный в виде DLL-библиотек (имя файла wsock32.dll для 32-битовых Windows). В случае программирования на языках высокого уровня каждый вызов WinSock-функций удобно обрамлять ("обертывать") соответствующей функцией языка; например функции send() посылки заполненного буфера и recv() приема данных в буфер удобно оформить таким образом:

```
procedure _TSocket.SocketSend(SendSocket:  _SOCKET;  hWnd: THandle;
            wMsg: Word; sSendStr: String;  NewState: Integer);
var  liBytesSent: Integer;
begin
 ii_SocketState := NewState;
 StrDispose(icp_Buffer);
 icp_Buffer := StrAlloc(1024);
 StrPCopy(icp_Buffer, sSendStr);
 WSAAsyncSelectfSendSocket, hWnd, wMsg, FD_READ+FD_WRITE);
 liBytesSent := send(SendSocket,  icp_Buffer,  StrLen(icp_Buffer),  0);
 if (liBytesSent = SOCKET_ERROR) then
   if (WSAGetLastError <> WSAEWOULDBLOCK) then
    ShowMessage(SocketError);
end; { конец SocketSend }

procedure _TSocket.SocketReceive(RecvSocket: _SOCKET;  hWnd: THandle;
            wMsg: Word; sBuffer: PChar; BufSize: Integer; NewState: Integer);
var  liBytesReceived: Integer;
begin
 WSAAsyncSelect(RecvSocket, hWnd, wMsg, FD_READ+FD_WRITE);
```

```
 liBytesReceived := recv(RecvSocket, sBuffer, BufSize, 0);
 if (liBytesReceived = SOCKET_ERROR) then
   if (WSAGetLastError <> WSAEWOULDBLOCK) then
     ShowMessage(SocketError);
 ii_SocketState := NewState;
end; { конец SocketReceive }
```

Таким образом процедуры-“обертки” инкапсулируют необходимые для функционирования WinSock-вызовов локальные переменные и обрабатывают ошибки вызовов библиотеки WinSock API (большинство из них возвращают константу SOCKET_ERROR при ошибке, выдача сообщений осуществляется посредством пары функций WSAGetLastError из WinSock API и ShowMessage из Windows API).

Для создания полноценного сетевого приложения необходимы функции InitializeSocket (инициализация интерфейса WinSock), ShutdownSocket (завершение работы WinSock), GetHostAddress (поиск IP-адреса целевого компьютера по заданному имени), ProcessHostAddress (проверки корректности полученного с помощью GetHostAddress адреса), GetHostAddressString (преобразование адреса в строку с необходимым изменением порядка следования байт в адресе), CreateSocket (создание сокета как объекта _SOCKET), CloseSocketConnection (отсоединение приложения от соединения с удаленным компьютером). Функция более высокого уровня OpenSocketConnection использует вышеописанные для создания и настройки сокета, перевода его в асинхронный режим и соединения с удаленным компьютером. Эти функции описаны в файлах TSocketc.pas (описание абстрактного класса TSocket) и NWinSock.pas (определение Pascal-интерфейса к WinSock API), доступны и C-варианты функций.

Заметим, что определение IP-адреса удаленного компьютера с помощью (использующей системный вызов библиотеки WinSock API функции WSAAsyncGetHostByName) процедуры-'обертки' GetHostAddress совместно с функцией ProcessHostAddress в зависимости от настройки локальной сети и клиентского компьютера может может оказаться безрезультатным; в этом случае следует использовать IP-адреса.

Для проверки функционирования сокета (состояния сети, своего компьютера и др.) удобно использовать стандартные порты (16-разрядный номер порта однозначно идентифицирует службу сети):

- **Порт Echo (порт 7).** Возвращает приложению клиента то, что он принял от клиента. Для проверки использования этого порта удаленной машине-серверу (host-компьютеру) посылается строка символов (содержимое строки не имеет значения, возможна случайная комбинация символов) и затем принимается строку ответа от host-компьютера. После приема законченной строки от host-компьютера она сравнивается с посланной строкой. Если

они идентичны – программа (реализация гнезда) работает правильно. Если две строки не согласуются, следует проверить корректность программу.

- **Порт отбрасывания (порт 9).** Используется для проверки только соединения и отправления. При соединении с таким портом все посылаемые строки отбрасываются (приема данных от host-компьютера нет). Эта служба узла аналогична нуль-устройству UNIX.
- **Порт Telnet (порт 23).** Используется для выполнения диалоговых сеансов с терминалами. При соединении с портом Telnet приложение клиента должно получить зарегистрированную подсказку. Если имеется зарегистрированный идентификатор подсказки и пароль для использования сервера, с которым произошло соединение, можно выдавать несколько команд и наблюдать результаты, которые вернутся к приложению-клиенту.
- **Порт сервера времени (порт 37).** При соединении клиента с портом времени он сразу выполняет операцию. Порт посылает компьютеру клиента текущее системное время и после этого отсоединяется. Для анализа и выдачи пользователю системного времени необходимо преобразовать его в строку (**служба порта 13** возвращает текущую дату и время).

Информация об этих (и многих других) службах находится в файле SERVICES (этот файл обычно располагается в каталоге C:\WINDOWS\SYSTEM32\DRIVERS\ETC\); совместно с файлом PROTOCOL этого достаточно для доставки данных корректному приложению.

Разрабатываемое в рамках данной работы приложение использует таблицу состояний, каждое из которых определяет режим его работы (переменные с с именами, начинающимися с SKT_):

1. Нет операции.
2. Получение имени host-компьютера (удаленной машины-сервера, на которой работает служба времени).
3. Соединение с host-компьютером.
4. Отправление сообщения host-компьютеру.
5. Прием сообщения от host-компьютера.

В файле Socket.pas объявлены 5 констант, соответствующих этим состояниям и идентифицирующее пользовательское событие константа UWM_SOCKETEVENT (это событие возникает, когда вызываемая из GetHostAddress асинхронная функция WSAAsyncGetHostByName закончит свою работу):

```
const
  SKT_NOOP = 0;
```

```
  SKT_GETHOSTADDR = 1;
  SKT_CONNECTTOHOST = 2;
  SKT_SENDPACKET = 3;
  SKT_RECVPACKET = 4;

  UWM_SOCKETEVENT = WM_USER + 500;
```

Добиться упрощения программы можно путем отказа от вызова функции WSAAsyncGetHostByName, однако этот вариант неприемлем при работе в многопотоковом режиме (когда каждое приложение выполняется в своем потоке, полностью не зависящем от интерфейса пользователя). Задающая подсоединение к порту 13 сервера службы времени константа IPPORT_DAYTIME=13 определена в файле NwinSock.pas.

Процедура UWMSocketevent проверяет текущее состояние гнезда и вызывает соответствующую функцию обработки состояния (процедура UWMSocketevent объявлена как

```
procedure UWMSocketEvent(var Msg:TMessage); message UWM_SOCKETEVENT;
```

т.е. является автоматически вызываемой при совершении события UWM_SOCKETEVENT):

```
procedure TForm1.UWMSocketEvent (var Msg: TMessage);
begin
  case NetSocket.GetSocketState of
    SKT_NOOP                : ; { ничего делать не надо }
    SKT_GETHOSTADDR     : ProcessHostLookup;
    SKT_CONNECTTOHOST : ProcessHostConnect ;
    SKT_SENDPACKET       : ProcessHostSend;
    SKT_RECVPACKET       : ProcessHostReceive;
  end;
    end; { конец  UWMSocketEvent }
```

Подобный прием (создание программ, управляемых пользовательскими событиями) является типичным при программировании сетевых приложений; процедура UWMSocketEvent фактически управляет программой.

При щелчке на кнопке Connect справа на главном окне, рис.10 процедура ConnectClick() проверяет, введено ли пользователем имя host-компьютера в текстовом окне Host Name и инициализирует процесс поиска адреса host-компьютера. Далее вызывается (ранее описанная) функция GetHostAddress (после окончания поиска IP-адреса возникает событие UWM_SOCKETEVENT, которое инициирует процедуру UWMSocketEvent, управляющую программой):

```
procedure TForm1.ConnectClick(Sender: TObject);
```

```
begin
 if Length(HostName.text) > 0 then { адрес host-машины введен… }
 begin
   HostResponse.Clear; { очистка Response (тип Tmemo ) }
   HostResponse.Lines.Add ('Looking Up Host Address'); { вывод в Response }
   NetSocket.GetHostAddress (Handle,
                UWM_SOCKETEVENT, { при окончании возбудить это событие }
                HostName.text, SKT_GETHOSTADDR);
 end
 else
   HostResponse.Lines.Add('Input InterNet Address, Please...');
end; { конец ConnectClick }
```

Процедура ProcessHostLookup индицирует состояние поиска, выводит найденный IP-адрес (в виде строки) в поле Address, создает сокет и соединяет его с портом сервера времени:

```
procedure TForm1.ProcessHostLookup;
begin
if (NetSocket.ProcessHostAddress = TRUE) then { удачно определен IP-адрес }
  begin
   HostResponse.Lines.Add('Attempt Connecting To Host');
   HostAddress.text := NetSocket.GetHostAddressString; { вывод IP в Address }
   isConnectionSocket := NetSocket.OpenSocketConnection(Handle,
                    UWM_SOCKETEVENT, { при окончании возбудить это событие }
                    SKT_CONNECTTOHOST,
                    IPPORT_TIMESERVER);
  end
  else
    HostResponse.Lines.Add('Cannot Find Host Address');
end; { конец ProcessHostLookup }
```

После соединения с host-компьютером ничего посылать ему не нужно; следует индицировать состояние программы и отправить сообщение о событии гнезда:

```
procedure TForm1.ProcessHostConnect;
var IsErrorString: String;
begin
  HostResponse.Lines.Add('Sending To Host');
  NetSocket.SetSocketState(SKT_SENDPACKET);
  ProcessHostSend;
end; { конец ProcessHostConnect }
```

Также настраивается программа на получения любых поступающих пакетов:

```
procedure TForm1.ProcessHostSend;
var  liBytesReceived: Integer;
begin
  HostResponse.Lines.Add('Receiving From Host');
  NetSocket.SetSocketState (SKT_RECVPACKET);
end; { конец  ProcessHostSend }
```

При приеме пакета необходимо убедиться в том, что он не пуст (host-компьютер часто сначала посылает пустой пакет), при получении пустого пакета запрос необходимо повторить. При приеме действительного пакета следует использовать функцию ntohl() для преобразования разрядов из порядка разрядов InterNet в порядок разрядов Intel (в исходно разработанном для UNIX стандарте последовательность байт в полном IP-адресе суть “1-2-3-4”, а для 32-битовых Intel-процессоров последовательность “2-1-4-3”) и закрыть гнездо:

```
procedure TForm1.ProcessHostReceive;
var IsRetnStr: PChar;
     llINetTime, llIntelTime: LongInt;
begin
{ время принимаем как длинное целое ilInetTime (объявлено глобальным) }
 recv(isConnectionSocket, ilINetTime, sizeof(LongInt), 0);
 if (ilINetTime <> 0) then { если принятое значение не пусто… }
 begin
   llIntelTime := ntohl(ilINetTime);
   IsRetnStr := InTime (llIntelTime); { перевод длинного целого в строку }
   IsRetnStr[(StrLen(IsRetnStr)-1)] := #0;
   HostResponse.Lines.Add (StrPas (IsRetnStr));
   NetSocket.SetSocketState (SKT_NOOP);
   NetSocket.CloseSocketConnection (isConnectionSocket);
 end;
end; { конец ProcessHostReceive }
```

Функция, преобразующая системное время host-компьютера в закрытую нулем строку, написана на C и оформлена как DLL с именем intimec.dll:

```
char *far PASCAL  _export  InTime (long inettime)
{ long llTime;
  char  *lsTimeStr;
 llTime = inettime - DATE_TIME_REF_POINT;
 lsTimeStr = ctime ((time_t  *) &llTime);
 return (lsTimeStr) ;
} /* конец  InTime */
```

Функция ctime() выполняет преобразование полученной метки времени (часто посылается в виде среднего времени по Гринвичу - GMT) в местное

время на основе временного пояса, который определен в конфигурации операционной системы клиентской машины.

Т.о. рассмотрен процесс создания простого клиентского приложения на основе сокетов. Единственным отличием между созданием приложения клиента WinSock и сервера WinSock является способ, используемый для соединения сокета с другим компьютером.

При создании первоначального гнезда (сокета) сервер должен вызвать функцию listen() для проверки запросов, которые ожидают соединения. Для обслуживания первого запроса, ожидающего очереди, используется функция accept().

Функция accept() создает новое гнездо с такими же характеристиками, как и у гнезда прослушивания. Необходимо соединить это гнездо с гнездом клиента и вернуть исходное гнездо в состояние прослушивания (это позволяет приложениям сервера обрабатывать многочисленные соединения гнезд одновременно). В многопотоковой среде приложение сервера часто отключается от нового потока для обработки "беседы" при каждом подсоединении гнезда клиента.

Необходимое оборудование – IBM PC-совместимая ЭВМ, предустановленная ОС Windows, пакет Delphi версии выше 2, утилита Telnet версии не ниже 1.0, стандартный броузер MS Explorer.

Порядок проведения работы. Студент проверяет функционирование сети посредством вызова службы времени через Telnet и штатный броузер, знакомится с правилами построения клиентских сокетных приложений, собирает клиентское приложение для подключения к службе времени, проверяет функционирование полученной системы.

Часть 1 работы. На этом этапе проверяется функционирование сетевого обеспечения клиентского компьютера. Некоторые адреса служб времени организации *National Institute of Standards and Technology* (NIST, www.nist.gov) приведены в табл.2.

Таблица 2.— WEB-адреса служб времени.

Name-адрес	*IP-адрес*	*Расположение*
time-a.nist.gov	129.6.15.28	NIST, Gaithersburg, Maryland
time-b.nist.gov	129.6.15.29	NIST, Gaithersburg, Maryland
time-a.timefreq.bldrdoc.gov	132.163.4.101	NIST, Boulder, Colorado
time-b.timefreq.bldrdoc.gov	132.163.4.102	NIST, Boulder, Colorado
time-c.timefreq.bldrdoc.gov	132.163.4.103	ST, Boulder, Colorado
utcnist.colorado.edu	128.138.140.44	University of Colorado, Boulder

time.nist.gov	192.43.244.18	NCAR, Boulder, Colorado
time-nw.nist.gov	131.107.1.10	Microsoft, Redmond, Washington
nist1.symmetricom.com	69.25.96.13	Symmetricom, San Jose, California
nist1-dc.glassey.com	216.200.93.8	Abovenet, Virginia
nist1-ny.glassey.com	208.184.49.9	Abovenet, New York City
nist1-sj.glassey.com	207.126.98.204	Abovenet, San Jose, California
nist1.aol-ca.truetime.com	207.200.81.113	TrueTime, AOL facility, Sunnyvale, California
nist1.aol-va.truetime.com	64.236.96.53	TrueTime, AOL facility, Virginia

На рис.7 приведен вид окна утилиты Telnet с одним из адресов службы времени (в зависимости от настройки сетевого ПО возможен ввод Name- или IP-базированных адресов), на рис.8 – то же окно после приема текущей даты и времени (следует обратить внимание, что после приема данных соединение сразу закрывается).

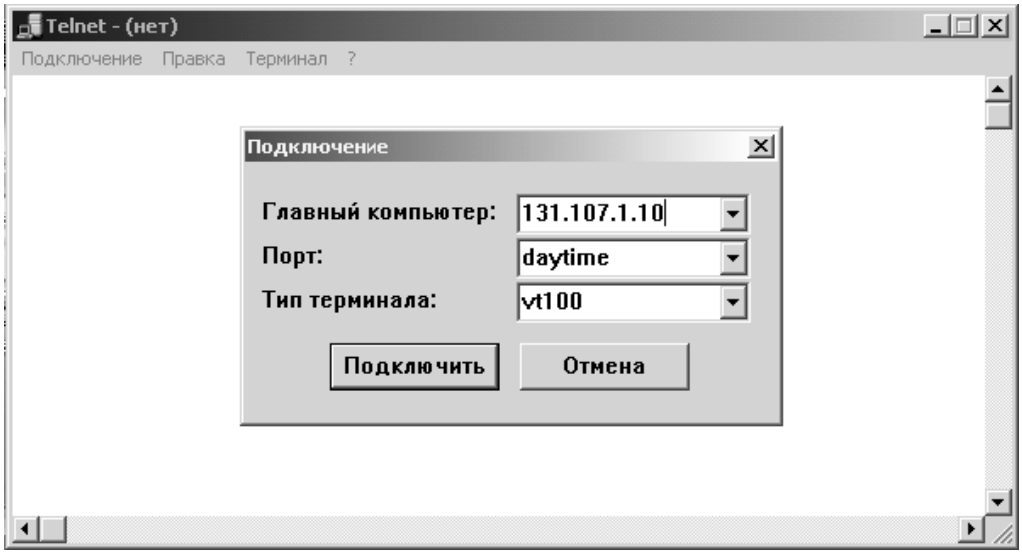

Рисунок 7.— Использование Telnet для подключения к порту службы времени

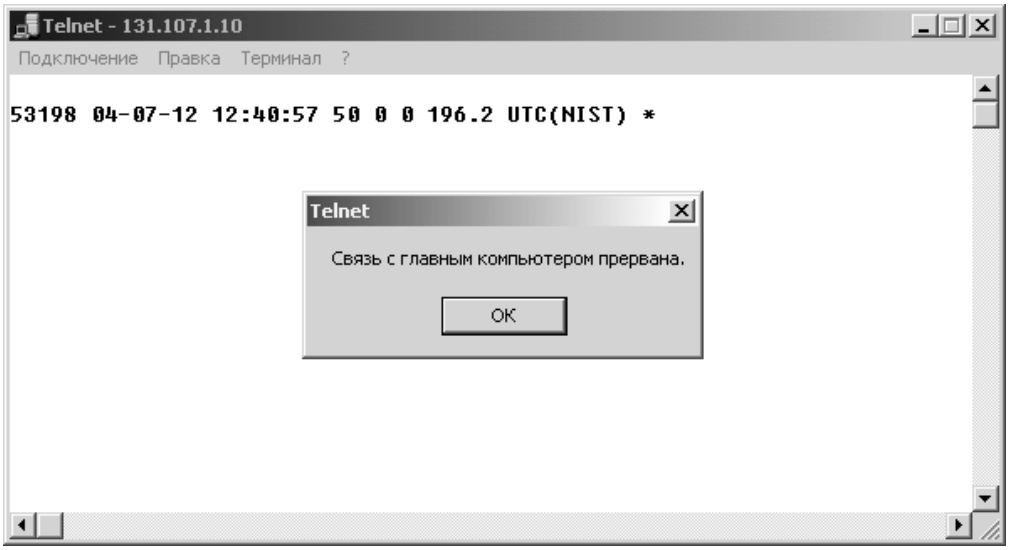

Рисунок 8.— Состояние окна Telnet после приема текущей даты и времени

На рис.9 показано использование штатного броузера, используемого для приема даты и времени (используется адрес http://time-nw.nist.gov:13 , где цифра 13 означает номер порта службы времени).

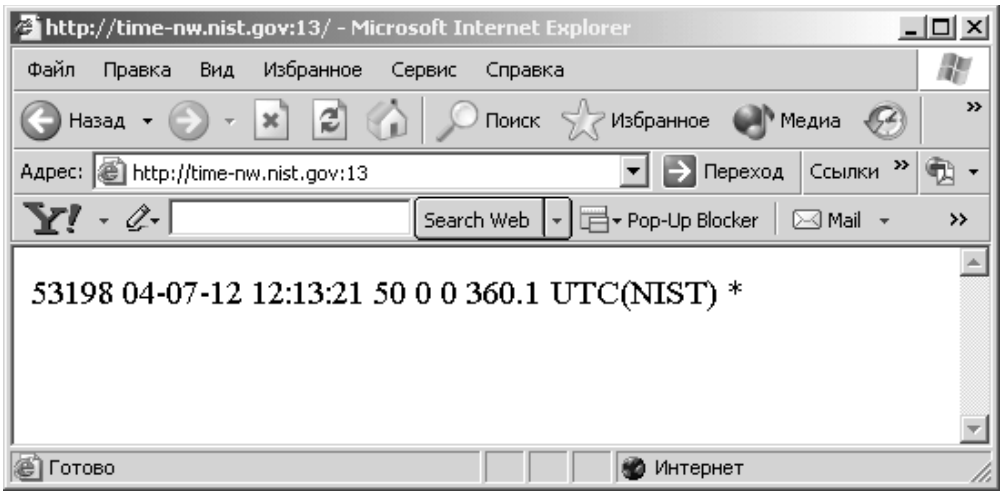

Рисунок 8.— Использование штатного броузера для подключения к порту службы времени

Часть 2 работы. На этом этапе с использованием системы Delphi создается действующее приложение службы времени. Для постройки приложения необходимы файлы Socktime.dpr, Socktime.dof (проект приложения), Socket.pas и Socket.dfm (главный файл), Tsocketc.pas и NWinSock.pas (вспомогательные файлы) и файл Intime32.dll. Путем анализа исходных тексты изучаются приемы программирования с использованием сокетов, компилируется исполняемый файл приложения Socktime.exe, проверяется его работоспособность (рис.10).

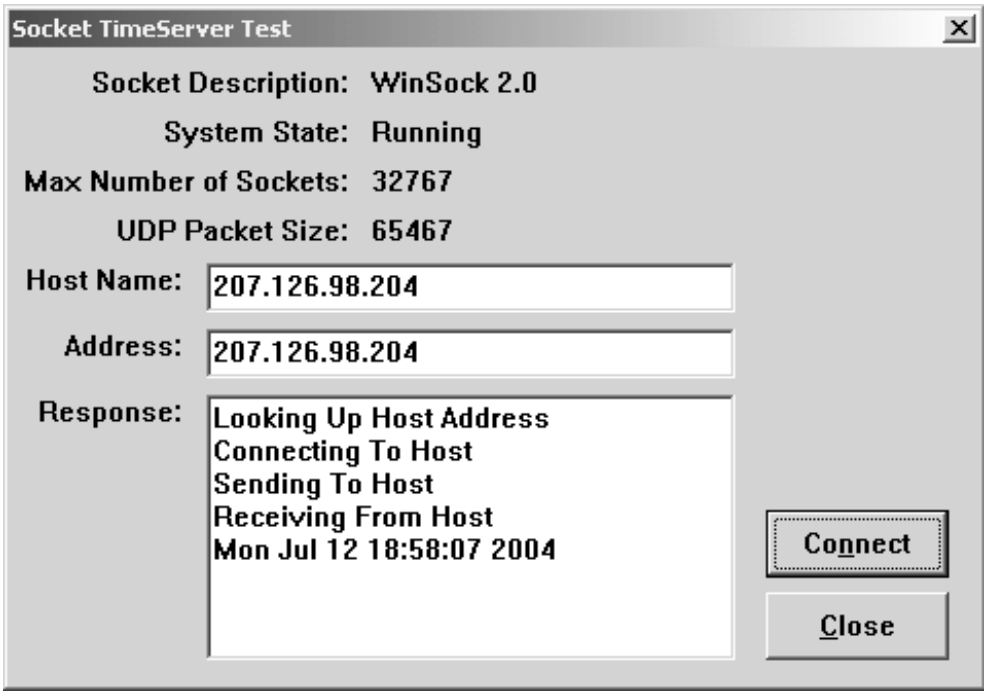

Рисунок 10.— Окно приложения для проверки работоспособности гнезда путем подключения к порту службы времени

По заданию преподавателя приложение может быть целенаправленно изменено, например:

- Переработан (в сторону повышения удобства) пользовательский интерфейс программы.
- Введена дополнительная проверка корректности вводимого типа сетевого адреса (InterNet или IP-адрес).
- Возможность изменения номеров портов служб (например, программирование посылки строки с ее приемом и сравнением с исходной для Echo-порта удаленной машины).

Темой (домашнего) задания повышенной сложности может быть переработка исходных текстов в нотацию языка C/C++ под C++Builder.

Альтернативой программированию на уровне сокетов является использование объектов (в смысле объектно-ориентированного программирования - ООП), инкапсулирующих нужный код и данные (например, входящие в комплект поставки Delphi и C++Builder старших версий объекты типа TNMTime и TNMDayTime разработки фирмы NetMasters L.L.C, www.netmastersllc.com, обеспечивающие прием данных от служб времени и даты-времени по портам 37 и 13 соответственно, см. рис.11); разработок подобных объектов известно великое множество.

Рисунок 11.— Окна приложений TimeDemo.exe и DtmDemo.exe, созданных с использованием объектов TNMTime и TNMDayTime

Оформление отчета по работе. В отчете приводятся список необходимых для реализации заданной функциональности программы стандартных функций библиотеки WinSock API, диаграмма состояний клиентской программы связи со службой порта времени, альтернативные возможности получения данных с рассматриваемых служб.

Вопросы для самопроверки.

1. Назвать стандартные службы, реализованные в InterNet.
2. Что такое порт и с какой целью введено это понятие?
3. Какими уникальными сочетаниями параметров определяется настройка сокета?
4. В чем заключается концепция использования функций-‘оберток’ при программировании сетевых приложений с использованием стандартных функций библиотеки WinSock API?

5. *Лабораторная работа № 5.* Разработка серверного сетевого приложения на основе сокетов (однопотоковый WEB-сервер)

Цель работы – приобретение практических знаний и навыков в разработке сетевых программ, использующих технологию сокетов (логических гнезд).

Теоретическая часть. При обмене информацией между клиентским приложением (броузером) и WEB-сервером используется протокол HTTP (*HyperText Transfer Protocol*).

Простой запрос по HTTP 0.9 состоит только из команды и имени ресурса; сервер в ответ направляет клиенту запрашиваемый ресурс (обычно HTML-файл) или сообщение о невозможности обслуживания. Большинство современных броузеров поддерживают обладающий многими усовершенствованиями формат HTTP 1.1 (описан в RFC-2068), ведутся работы по внедрению формата HTTP NG (*New Generation*); эти форматы допускают запросы нескольких файлов при одном подсоединении к серверу, благодаря чему существенно увеличивается скорость обмена информацией [2,5].

Полные форматы запросов и ответа для HTTP 1.0 являются многострочными сообщениями, которые заканчиваются пустой строкой. Эти заголовки сообщений информируют друг друга о версии HTTP, посредством которой каждый из них может поддерживать связь, о размере и дате запрашиваемого ресурса, о типе ресурса, который клиент может принять, а также о командах, поддерживаемых сервером и др. Согласно спецификации HTTP 1.0 запросы и ответы HTTP оформляются в формате сообщения *Internet Message Format.*

Полный запрос (HTTP 1.0) состоит минимум из одной строки заголовка HTTP и строки запроса. После строки запроса могут следовать строки общего заголовка, строки заголовка запроса, строки заголовка сущности, а в конце обязательно пустая строка. Типичный запрос выглядит следующим образом (каждая строка завершается кодами “Возврат каретки” и “Перевод строки” - #13#10, комментарии выделены символами { } согласно Pascal-нотации);

```
{ Команда (метод) GET, запрашивающая стартовую страницу сайта (обычно
  это INDEX.HTML, INDEX.HTM или DEFAULT.HTM). Однако можно запросить
  и произвольный файл: GET имя_файла HTTP/1.0#13#10 }
GET / HTTP/1.0#13#10
{ Команда Pragma, предписывающая выдать данные по этому
  запросу без кэширования (т.н. “актуальные данные”) }
Pragma: no-cache#13#10
{ Клиент хочет принимать массив битов как изображения GIF и/или JPEG }
Accept: image/x-bitmap, image/gif, image/jpeg#13#10
```

```
{ Текущая дата машины клиента }
Date: Wed, Feb 7, 1996 21:30 CST#13#10
{ Клиентское ПО совместимо с MIME 1.0 }
MIME-Version: 1.0#13#10
{ E-Mail пользователя, запрашивающего данные }
From: e881@yahoo.com#13#10
{ InterNet-адрес, из которого был взят данный URL }
Referer: http://pilger.mgapi.edu#13#10
{ Запрашивающее приложение на клиентской машине }
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)#13#10
#13#10 { в конце обязательна пустая строка }
```

Согласно спецификации HTTP 1.1 список команд дополнен еще примерно 10 методами (OPTIONS, PUT, PATCH, COPY, MOVE, DELETE, LINK, UNLINK, TRACE, WRAPPED). Более подробную информацию о различных заголовках для использования в запросе можно найти по адресу www.w3c.org.

Формат строки полного ответа сервера близок формату полного запроса. Различие между ними состоит в формате первой строки, строки статуса и типах информации заголовка сообщения, которые, возможно, будут возвращены. Формат строки статуса предназначен для того, чтобы выдать сначала информацию о версии HTTP с последующим числовым кодом ответа, а затем некоторый текст с текстовым пояснением причины возврата этого кода. Типовой полный ответ имеет примерно следующий вид:

```
{ Это положительный ответ на полный запрос, выданный клиентом
  (см. выше) - сервер отвечает “OK” . Самое распространенное сообщение об
  ошибке: HTTP/1.1 404 Not Found – указанная страница не найдена на сервере }
HTTP/1.0 200 OK#13#10
{ Текущее время системы на сервере }
Date: Thu, 8 Feb 1996 20:06:43 GMT#13#10
{ Совместимость с MIME следующей страницы (строка заголовка иногда
  используется неправильно в заголовках сообщений HTTP }
MIME-Version: 1.0#13#10
{ Приложение сервера }
Server Delphi Web Server vl.0#13#10
{ Список команд, поддерживаемых сервером }
Allow: GET, HEAD#13#10
{ Запрашиваемая страница имеет длину 2323 байта }
Content-Length: 2323#13#10
{ Запрашиваемая страница является документом HTML }
Content-Type: text/html#13#10
#13#10
```

После этого заголовка сразу же следует запрашиваемый ресурс (чаще всего это файл HTML, но может быть и (двоичный) файл текста, мультимедиаданных и др.).

При возврате ресурса клиенту сначала сервер ищет запрашиваемый файл, причем имя его берется из поля GET и рассматривается *относительно корневого каталога сайта* (этот каталог указывается при конфигурировании сервера). Запрашиваемый файл не может находиться выше (по файловой системе) текущего каталога вследствие требования защищенности WEB-сервера.

Если файл существует и доступен для чтения, сервер исследует различную информацию о нем (для последующего сообщения клиенту, что будет передано) и начинает передачу файла через гнездо (в случае команды HEAD сервер ограничивается передачей информации о файле). При использовании (вышеописанных) технологий CGI- или ISAPI активизируется соответствующее расширение сервера и возвращается запрошенный скрипт**о**м ресурс.

Ограничения разрабатываемого WEB-севера:

- Используются протоколы HTTP 0.9 (простой) и HTTP 1.0 (полный).
- Обслуживается только одного соединение с клиентом в каждый момент времени.
- Отсутствует поддержка GCI- и ISAPI-расширений сервера.

При запуске WEB-сервер выполняет следующие действия:

1. Создает гнездо на HTTP TCP-порту (порт 80 по умолчанию) и начинает прослушивание поступающих запросов на подсоединение.
2. Принимает любые запросы на подсоединение.
3. Если некоторое соединение уже активно, отправляет новому соединению ответ о неготовности и отсоединяет клиента. Если нет уже существующих соединений, ждет отправления запроса клиентом.
4. Анализирует принятые клиентские запросы для определения существования и готовности к передаче запрашиваемого файла.
5. Возвращает запрашиваемый файл клиенту.
6. Отсоединяется от клиента и ожидает следующего запроса на подсоединение.

Переменная IbIsFullRequest: Boolean; определяет версию протокола HTTP и устанавливается в зависимости от того, какой запрос послал клиент (FALSE для HTTP 0.9 и TRUE при HTTP 1.0). Вызов функции для создания гнезда прослушивания выглядит следующим образом:

```
{ создать гнездо прослушивания на порту 80 TCP }
isListenSocket := NetSocket.CreateListenSocket(handle,
                        WM_LISTENSOCKETEVENT, 80);
```

Процедура ReadCommandMsg считывает в цикле принятое от клиента сообщение:

```
procedure TWbServe.ReadCommandMsg;
var IsCommand: String; IsMessage: String;
     liCmdbength: Integer; liMsgLength: Integer;
begin
{ Получить запрос от клиента с проверкой ошибок }
  liMsgLength := recv(isConnectionSocket, ReceiveBuffer, 8192, 0);
  if (liMsgLength = SOCKET_ERROR) and
                        (WSAGetLastError <> WSAEWOOLDBLOCK) then
    ShowMessage(NetSocket.SocketError) { выдать сообщение об ошибке }
 else
 begin
{ Закрыть принятую строку запроса нулем }
   ReceiveBuffer[liMsgLength] := Char(0);
{ Получить первую строку принятого запроса }
  if NetSocket.GetLine(TempBuffer, ReceiveBuffer) then
  begin
{ Выдать принятую строку на экран для пользователя }
   mleReceived.Lines.Add(StrPas(TempBuffer));
{ Есть еще ли что-нибудь в буфере захвата? }
   if (Length(isHoldBuffer) > 0) then
{ Если да, добавьте его перед новой строкой }
  IsMessage := isHoldBuffer + StrPas(TempBuffer)
  else
   IsMessage := StrPas(TempBuffer);
{ Очистить буфер захвата }
   isHoldBuffer := '';
{ Найти символ пробела для определения конца команды }
   liCmdLength := Pos(' ', IsMessage);
   if (liCmdLength = 0) then
   begin
{ Если нет символа пробела, это не командная строка заголовка }
    liCmdLength := Length(IsMessage);
    IsCommand := IsMessage;
    IsMessage := '';
   end
   else
```

```
   begin
{ Отделить команду от остальной части запроса }
    IsCommand := Copy(IsMessage, 1, (liCmdLength - 1));
    IsMessage := Copy(IsMessage, (liCmdLength + 1), Length(IsMessage));
   end;
{ Передать команду и параметр процедуре CheckCommandMsg }
   CheckCommandMsg(IsCommand, IsMessage);
   end;
{ Любые дополнительные строки не анализируются в этом
  приложении, поэтому просто вывести их на экран }
  while NetSocket.GetLine(TempBuffer, ReceiveBuffer) do
  begin
   mleReceived.Lines.Add(StrPas(TempBuffer));
  end;
{ Осталось ли что-нибудь? Если да, поместить его в буфер захвата }
  if (Length(StrPas(TempBuffer)) > 0) then
  begin
   if (Length(isHoldBuffer) > 0) then
   isHoldBuffer := isHoldBuffer + StrPas(TempBuffer)
  else
   isHoldBuffer := StrPas (TempBuffer) ,
  end;
 end;
end; { конец ReadCommandMsg }
```

После приема команды и ее аргументов необходимо определить, есть ли в списке маркер HTTP 1.0 (иначе работаем с HTTP 0.9). Процедура CheckCommandMsg проверяет, какая команда получена сервером:

```
procedure TWbServe.CheckCommandMsg(asCommand, asArguments: String);
var IbCmdFound: Boolean; liSpacePos: Integer;
begin
{ Инициализировать флаг обнаруженной команды значением FALSE }
 IbCmdFound := FALSE;
{ Найден ли индикатор HTTP 1.0  ? }
 if (Pos('HTTP/1.0', asArguments) > 0) then
 begin
{ Да, отметить этот запрос как полный (HTTP 1.0) }
 iblsFullRequest := TRUE;
{ Теперь удалить индикатор HTTP 1.0 из имени файла)
 liSpacePos := Pos(' ', asArguments);
 if (liSpacePos > 0) then
  isFileName := Copy(asArguments, 1, (liSpacePos — 1));
 end
```

```
 else
 begin
{ Нет индикатора HTTP 1.0, поэтому необходимо
  обрабатывать этот запрос как HTTP 0.9 }
 iblsFullRequest := FALSE;
 isFileName := asArguments;
 end;
{ Была запрошена страница по умолчанию ? }
 if (isFileName = '/') then
  isFileName := sleDefPage.Text
 else
{ Нет, добавить корневой каталог в запрашиваемое имя файла }
 isFileName := sleDefDir.Text + isFileName;
{ Вызвать процедуру ConvertFileNameToDos для преобразования
 имени файла в DOS-формат, чтобы легче его использовать)
 ConvertFileNameToDos;
 if (asCommand = 'GET') then { Это команда GET ? }
 begin
{ Это полный или короткий запрос (HTTP 1.0 или HTTP 0.9) ? }
 IbCmdFound := TRUE;
 iiHTTPStatus := http_GET;
 if (iblsFullRequest) then
  SendFullObject
 else
  SendShortObject;
 end;
if (asCommand = 'HEAD') then { Это команда HEAD ? }
 begin
{ Это полный или короткий запрос (HTTP 1.0 или HTTP 0.9) ? }
  IbCmdFound := TRUE;
  iiHTTPStatus := HTTP_HEAD;
  if (iblsFullRequest) then
   SendFullHeader
  else
   SendShortHeader;
  end;
 if (asCommand = 'POST') then { Это команда POST ? }
 begin
{ Да, отправить клиенту ответ NOT SUPPORTED (не поддерживается) }
  IbCmdFound := TRUE;
  iiHTTPStatus := HTTP_POST;
  SendContextMsg;
  CloseConnection; { Закрыть соединение }
 end;
 if (not IbCmdFound) then { Не нашли допустимую команду… }
```

```
 begin
{ Отправить ответ COMMAND UNKNOWN (неизвестная команда) }
  iiHTTPStatus := HTTP_UNKWN;
  SendContextMsg;
  CloseConnection; { Закрыть соединение }
 end;
end; { конец CheckCommandMsg }
```

Если запрос пришел от клиента HTTP 0.9, сервер информирует, что команда HEAD не допускается:

```
procedure TWbServe.SendShortHeader;
var isShortHeaderMsg: String; begin
{ создать HTML-документ, сообщающий клиенту, что команда HEAD
 недопустима в протоколе HTTP 0.9 }
 isShortHeaderMsg := '<TITLE>Operation Not Allowed</TITLE>';
 { отправить это сообщение клиенту }
 NetSocket.SocketSend(isConnectionSocket, handle, UWM_SOCKETEVENT,
                      isShortHeaderMsg, SKT_NOOP);
 mleSent.Lines.Add (isShortHeaderMsg); { выдать сообщение на экран }
 CloseConnection; { закрыть соединение }
end; { конец SendShortHeader }
```

Если клиент HTTP 0.9 выдает команду GET, сервер должен определить, существующий ли файл запрашивает клиент; в случае положительного ответа на этот вопрос файл отправляется клиенту, в противном случае возвращается короткое HTML-сообщение:

```
procedure TWbServe.SendShortObject;
var isShortObjectMsg: String;
begin
 if FileExists(isFileName) then { Запрашиваемый файл существует ? }
 begin
{ Открыть и отправить его клиенту }
 OpenSendFile;
 ProcessDataSend;
 end
 else
 begin
{ Нет, создать HTML-документ HTML с информацией об
 отсутствии запрашиваемого файла }
 isShortObjectMsg := '<TITLE>Not Found</TITLE>';
{ Отправить созданное сообщение клиенту }
 NetSocket.SocketSend(isConnectionSocket, handle,
```

```
                                UWM_SOCKETEVENT, isShortObjectMsg, SKT_NOOP);
 mleSent.Lines.Add(isShortObjectMsg); { Вывести его на экран пользователю }
 CloseConnection; { Закрыть соединение }
 end;
end; { конец SendShortObject }
```

Процедура SendFullHeader отправляет заголовок ответа поддерживающего протокол HTTP 1.0 клиенту, проверяя существование запрашиваемого файла и выдавая соответствующие ответы (код 200 или 404 при существующем / несуществующем файле соответственно):

```
procedure TWbServe.SendFullHeader;
begin
 if FileExists(isFileName) then { Запрашиваемый файл существует ? }
 begin
{ Открыть файл и отправить полный заголовок сообщения, передавая
 функцию FIleHeader в качестве дополнительного параметра строки
 сообщения процедуре SendFullMsg }
 OpenSendFile;
 SendFullMsg(isConnectionSocket, 200, HTTP_NOOP,
                  UWM_SOCKETEVENT, FileHeader);
 CloseConnection; { Закрыть соединение }
 end
 else
 begin { Файл не существует – послать клиенту сообщение об этом }
 SendFullMsg(isConnectionSocket, 404, HTTP_NOOP,
                    UWM_SOCKETEVENT, isFileName);
 CloseConnection; { Закрыть соединение }
 end;
end; { конец SendFullHeader }
```

При обработке команды GET нижеприведенная процедура SendFullObject отправляет клиенту не только данные о запрашиваемом файле, но и сам файл (вместо CloseConnection вызывается ProcessDataSend):

```
procedure TWbServe.SendFullObject;
begin
 if FileExists(isFileName) then { Запрашиваемый файл существует ? }
 begin
{ Открыть его и отправить полное сообщение заголовка, причем
  передать функцию FileHeader в качестве дополнительного параметра
  строки сообщения процедуре SendFullMsg }
 OpenSendFile;
```

```
 SendFullMsg(isConnectionSocket, 200, HTTP_NOOP,
                    UWM_SOCKETEVENT, FileHeader);
 ProcessDataSend; { Отправить файл }
 end
 else
 begin
{ Файла нет, отправить клиенту полное сообщение об этом }
 SendFullMsg(isConnectionSocket, 404, HTTP_NOOP,
                   UWM_SOCKETEVENT, isFileName);
 CloseConnection; { Закрыть соединение }
 end;
end; { конец SendFullObject }
```

При отправлении клиентом команды POST или другой не поддерживаемой сервером команды необходимо ответить соответствующим сообщением для информирования клиента об этом; для этого служит процедура SendContextMsg:

```
procedure TWbServe.SendContextMsg;
begin
{ Какую команду отправил клинт ? }
 case iiHTTPStatus of
 HTTP_NOOP: ; { HOOP, GET и HEAD - ничего делать не надо, команды поддер-
                    живаются сервером }
 HTTP_GET    : ;
 HTTP_HEAD  : ;
{ POST и неизвестная команда - отправить ответы 401 и 400 соответственно }
 HTTP_POST:      SendFullMsgfisConnectionSocket, 401, HTTP_NOOP,
                    UWM_SOCKETEVENT, '') ;
 HTTP_UNKWN: SendFullMsg(isConnectionSocket, 400, HTTP_NOOP,
                    UWM_SOCKETEVENT, '') ;
 end;
end; { конец SendContextMsg }
```

Прежде проверки наличия и возможности открытия любых запрашиваемых файлов необходимо единообразным образом преобразовать их имена, заменяя прямые слеши “/” на обратные “\” (предлагаемая процедура ConvertFileNameToDos упрощена и не обрабатывает “длинные” имена файлов):

```
procedure TWbServe.ConvertFileNameToDos;
var liSlashPos: Integer;
begin
```

```
liSlashPos :=Pos('/', isFileName); { Найти первый символ '/' }
{ Выполнять цикл столько раз, сколько имеется символов '/' }
 while (liSlashPos > 0) do
 begin
{ Скопировать имя файла, заменяя символ '/' символом '\' }
 isFileName := Copy(isFileName, 1, (liSlashPos - 1)) + '\' + Copy(isFileName,
                    (liSlashPos + 1), Length(isFileName));
 liSlashPos := Pos('/', isFileName);
 end;
end; { конец ConvertFileNameToDos }
```

Большинство отправляемых сервером клиенту сообщений содержат одну и ту же информацию в одинаковом формате, поэтому удобно упростить отправление и форматирование этой информации. Для этого служит простая процедуру SendFullMsg (кроме стандартных сообщений отправляются строки заголовка, даты и времени, версию MIME, реализации сервера, эти данные также выводятся пользователю на экран):

```
procedure TWbServe.SendFullMsg(aSendSock:_SOCKET;
                               iMsg, iNewState, aiEvent: Integer;
                               asAdditionalMsg: String);
var IsMsgStr: String;
 begin
 case iMsg of { Инициализировать сообщение с помощью указанного кода ответа }
 200: IsMsgStr := 'HTTP/1.0 200 OK' + #13#10;
 201: IsMsgStr := 'HTTP/1.0 201 Created' + #13#10;
 202: IsMsgStr := 'HTTP/1.0 202 Accepted' + #13#10;
 204: IsMsgStr := 'HTTP/1.0 204 No Content' + #13#10;
 301: IsMsgStr := 'HTTP/1.0 301 Moved Permanently' + #13#10;
 302: IsMsgStr := 'HTTP/1.0 302 Moved Temporarily' + #13#10;
 304: IsMsgStr := 'HTTP/1.0 304 Not Modified' + #13#10;
 400: IsMsgStr := 'HTTP/1.0 400 Bad Request' + #13#10;
 401: IsMsgStr := 'HTTP/1.0 401 Unauthorized' + #13#10;
 403: IsMsgStr := 'HTTP/1.0 403 Forbidden' + #13#10;
 404: IsMsgStr := 'HTTP/1.0 404 Not Found' + #13#10;
 500: IsMsgStr := 'HTTP/1.0 500 Internal Server Error' + #13#10;
 501: IsMsgStr := 'HTTP/1.0 501 Not Implemented' + #13#10;
 502: IsMsgStr := 'HTTF/1.0  502 Bad Gateway' + #13#10;
 503: IsMsgStr := 'HTTP/1.0 503 Service Unavailable' + #13#10;
 end;
{ Добавить текущую дату и время }
 IsMsgStr := IsMsgStr + 'Date: ' +
                FormatDateTime('ddd, d mmm yyyy hh:mm:ss "CST"', Now());
{ Отправить клиенту это и добавить строку  на экран для пользователя }
```

```
 NetSocket.SocketSend(aSendSock, handle, aiEvent, IsMsgStr, SKT_NOOP);
 mleSent.Lines.Add(IsMsgStr);
{ Создать две строки  для сообщения клиенту о ПО, обслуживающее сей запрос }
 IsMsgStr := 'MIME-Version: 1.0' + #13#10;
 IsMsgStr := IsMsgStr + 'Server: Delphi Web Server v1.O' + #13#10;
{ Отправить эти 2 строки и то, что было передано в качестве строк
  информации в дополнительном заголовке; не забыть добавить их на экран)
 NetSocket.SocketSend(aSendSock, handle, aiEvent,
                              IsMsgStr + asAdditionalMsg, SKT_NOOP);
 mleSent.Lines.Add (IsMsgStr + asAdditionalMsg) ;
{ Сбросить флаг статуса текущей команды до заданного значения состояния }
 iiHTTPStatus := iNewState;
end; { конец SendFullMsg }
```

Открытие и определение размера запрашиваемого файла осуществляется процедурой OpenSendFile:

```
procedure TWbServe.OpenSendFile;
begin
{ установить флаг FileOpen в TRUE }
 ibFileOpen := TRUE;
{ присвоить имя файла дескриптору файла }
 AssignFile(TransferFile, isFileName);
{ выполнить сброс и открыть файл для считывания }
 Reset(TransferFile);
{ Считать размер файла в переменную размеров файла }
 ilFileSize := FileSize (TransferFile);
end; { конец OpenSendFile }
```

После открытия файла можно форматировать всю информацию о файле, которую нужно отправить клиенту (функция FileSizeHeader выдает данные о размере, а FileTypeHeader – о типе файла, FileHeader собирает эти данные вместе):

```
function TWbServe.FileSizeHeader: String;
begin
{ Создать строку заголовка сообщения о размере файла }
 FileSizeHeader := 'Content-Length: ' + IntToStr(ilFileSize) + #13#10;
end; { конец FileSizeHeader }

function TWbServe.FileTypeHeader: String;
var IsFileExt: String; IsFileType: String; liPosPeriod: Integer;
begin
```

```
{ Указать тип файла (по умолчанию это текст HTML) }
 IsFileType := 'text/html';
{ Нати расширение в имени файла }
 liPosPeriod := Pos(' .', isFileName);
 if (liPosPeriod > 0) then
 begin
{ Это файл типа .GIF или .JPEG ? На самом деле допустимых
 типов файлов намного больше; в расширенную версию этой
 функции следует включить их все }
 IsFileExt := Copy(isFileName, (liPosPeriod + 1), 3) ;
 if ((IsFileExt = 'gif') or (IsFileExt = 'GIF')) then { это GIF-файл }
  IsFileType := 'image/gif';
 if ((IsFileExt = 'jpg') or (IsFileExt = 'JPG')) then { это JPG-файл }
  IsFileType := 'image/jpeg';
end;
{ Создать строку заголовка типа содержимого файла }
 FileTypeHeader := 'Content-Type: ' + IsFileType + #13#10;
end; { конец FileTypeHeader }

function TWbServe.FileHeader: String;
var IsHeaderStr: String;
begin
{ Вставить все три строки вместе в раздел заголовка сообщения }
 IsHeaderStr := 'Allow: GET, HEAD' + #13#10;
 IsHeaderStr := IsHeaderStr + FileSizeHeader + FileTypeHeader;
 FileHeader := IsHeaderStr;
end; { конец FileHeader }
```

Собственно отправка файла клиенту осуществляется процедурой ProcessDataSend (если файл чрезмерно велик, придется усложнить процедуру):

```
procedure TWbServe.ProcessDataSend;
var liCurByteCount: Integer; IcCurByte: Byte;
     IpTransferDataPointer: PChar;
begin
 if ibFileOpen then { Запрашиваемый файл открыт ? }
 begin
 liCurByteCount := 0; { Инициализировать счетчик байтов }
 if (not Eof(TransferFile)) then { Мы уже в конце файла?)
 begin
 ClearBuffer(TransferBuffer, 8192); { Нет, очистить буфер передачи }
{ Прочитать 4’096 байтов из файла в буфер }
 while ((liCurByteCount < 4096) and (not Eof(TransferFile))) do { цикл по байтам }
```

```
 begin
 Read(TransferFile, IcCurByte);
 TransferBuffer[liCurByteCount] := Char(IcCurByte);
 Inc(liCurByteCount);
 end;
{ Отправить клиенту содержимое буфера }
 NetSocket.SocketSendBuffer(isConnectionSocket, handle,
                OWM_SOCKETEVENT, TransferBuffer, liCurByteCount,
                SKT_SENDPACKET);
 end
 else
 CloseConnection; { Весь файл отправлен, закрыть соединение }
 end;
end; { конец ProcessDataSend }
```

Закрытие соединения осуществляет процедура CloseConnection (при этом на единицу уменьшается число текущих соединений – чтобы иметь возможность открыть новые соединения без превышения общего числа **о**ных):

```
procedure TWbServe.CloseConnection;
begin
{ перед закрытием гнездовое соединение необходимо завершить его }
 shutdown(isConnectionSocket, 2);
{ закрыть гнездовое соединение и уничтожить гнездо }
 NetSocket.CloseSocketConnection(isConnectionSocket);
if ibFileOpen then { если запрашиваемый файл открыт, закрыть его }
 begin
  CloseFile(TransferFile);
  ibFileOpen := FALSE;
 end;
{ уменьшить на единицу число текущих соединений для того,  чтобы в будущем
  можно было принять больше поступающих запросов на подключение }
 iiNumConnected := iiNumConnected -1;
end; { конец CloseConnection }
```

Приведенные выше процедуры и функции расположены в файле Wserve.pas, остальные процедуры WEB-сервера тривиальны и в основном направлены на поддержку Delphi-программирования.

Рассматриваемый сервер обслуживает только одно соединение в данный момент времени, не поддерживает современный протокол HTTP 1.1, в нем отсутствует поддержка технологий CGI и ISAPI расширения функциональности, средств безопасности и поддержки для аутентификации клиента и шифрования связи, обслуживает запросы на перекачку только ограниченного ко-

личества типов файлов, не имеет средств восстановления при сбоях и др. Разработка универсального ядра WEB-сервера является трудоемкой задачей, причем именно здесь отлично проявил себя “принцип открытого кода” – лучший WEB-сервер Apache (около 60% применений) совершенствуется именно таким способом.

Для простой пересылки файлов между удаленными компьютерами удобно также использовать высокоуровневые компоненты TClientSocket и TServerSocket, реализующие клиентский и серверный сокеты соответственно. Также можно использовать входящие в поставку Delphi компоненты вышеупомянутой фирмы NetMasters - TNMMsgServ и TNMMsg (серверная и клиентская части; компоненты предназначены для обмена текстовыми сообщениями), TNMStrmServ и TNMStrm (серверная и клиентская части; компоненты предназначены для обмена потоками данных) и TNMUDP (компонент, реализующий протокол UDP - *User Datagram Protocol*, см. RFC768).

Необходимое оборудование – IBM PC-совместимая ЭВМ, предустановленная ОС Windows, пакет Delphi версии выше 2, стандартный броузер MS Explorer.

Порядок проведения работы. Студент знакомится (путем анализа приведенного выше исходного текста приложения) с правилами построения серверных сокетных приложений, собирает приложение W_SERVER, проверяет его функционирование посредством штатного броузера.

Компиляция приложения W_server.exe проводится стандартными методами (используются файлы W_server.dpr, W_server.dof, Wserve.pas, Wserve.dfm, Tsocketc.pas). Проверка функциональности может быть проведена на локальной машине в offline-режиме (как иллюстрировано рис.12,13) или в сети (может потребоваться дополнение файла HOSTS).

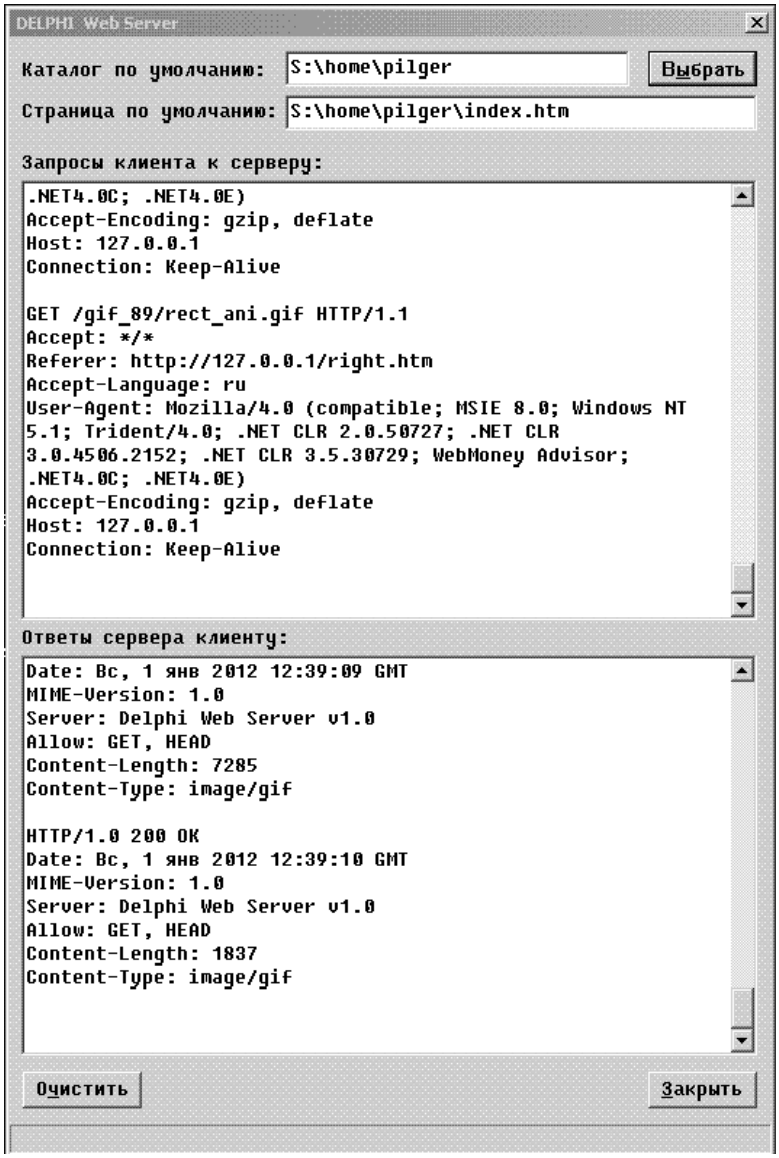

Рисунок 12.— Окно WEB-сервера, созданного на Delphi с использованием технологии сокетов

В вверхней строке ввода окна приложения WEB-сервера задается имя каталога сайта, в строке под ней – имя файла главной страницы (см. рис.12).

Видно (в третьем сверху поле вывода), что в данный момент сервером получен GET-запрос на передачу клиенту файла ./gif_89/rect_ani.gif по протоколу HTTP 1.0, текущий сайт 127.0.0.1 (адрес того же компьютера, где функционирует сервер), используется основанный на ядре Mozilla броузер MS IE 8.0 под Windows'NT 5.1 (Windows'XP).

В ответ на запрос сервер ответил сообщением об успехе доступа к файлу и его возврата (строка HTTP/1.0 200 OK в нижнем поле вывода), броузеру сообщается о высылке запрошенного GIF-файла длиной 1837 байт (такая информация будет обновляться для каждого запрашиваемого и отсылаемого файла, что является следствием использования именно HTTP 1.0, а не HTTP 1.1).

Интерпретация клиентским броузером высланной сервером страницы приведена на рис.13 и ничем не отличается от работы броузера в штатном режиме при просмотре сайта.

Следует отметить невысокую скорость отображения страницы, что является следствием пересылки протоколом HTTP 1.0 только одного файла при каждом подсоединении к серверу.

По заданию преподавателя приложение может быть доработано (домашнее задание повышенной сложности, см. RFC-2068, www.w3c.org, info.cern.ch, www.ics.uci.edu и [2,3,5]), например:

- Переработан (для повышения удобства) пользовательский интерфейс WEB-сервера.
- Обеспечение поддержки протокола HTTP 1.1.
- Введена поддержка технологий CGI- и ISAPI-расширений сервера.
- Включение многопоточности (обеспечение одновременного обслуживания нескольких клиентов).
- Введение встроенной поддержки баз данных (например, на основе предлагаемой Borland Int. технологии BDE/IDAPI).

Оформление отчета по работе. В отчете приводятся список необходимых для реализации заданной функциональности программы стандартных функций библиотеки WinSock API, диаграмма состояний программы. В отчете должна присутствовать функциональная часть содержимого файла HOSTS и несколько $(3 \div 5)$ заголовков запросов клиента серверу и (соответствующих) ответов WEB-сервера броузеру (с указанием местоположения принятых объектов в окне последнего).

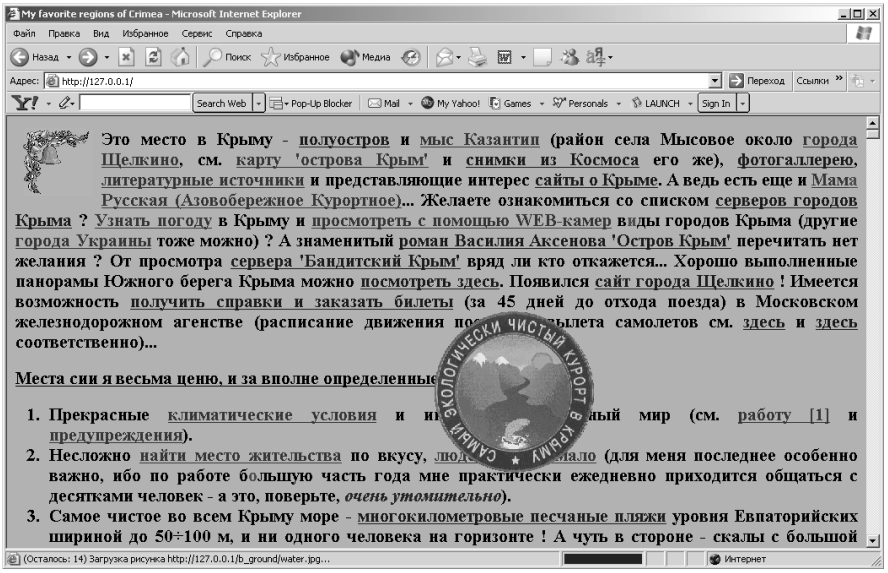

Рисунок 13.— WEB-страница, полученная с помощью созданного на Delphi по технологии сокетов WEB-сервера

Вопросы для самопроверки.

1. Каковы основные функции WEB-сервера?
2. Какова последовательность при обмене информацией между WEB-сервером и клиентским приложением?
3. В чем различия между протоколами HTTP 0.9, HTTP 1.0 и HTTP 1.1 ? Какие методы поддерживают эти протоколы? С какой целью разрабатывается протокол HTTP NG?
4. Каким образом клиентское приложение получает информацию о возможности выполнения своего запроса серверу?
5. Каким образом определяется требование пересылки клиенту двоичного файла?

6. *Лабораторная работа № 6.* Разработка клиенского сетевого приложения с использованием высокоуровневых компонентов (упрощенный броузер)

Цель работы – приобретение практических знаний и навыков в разработке сетевых клиентских программ, использующих технологию высокоуровневых объектов (компонентов) Delphi / C++Builder.

Теоретическая часть. Создание коммерческого WEB-броузера является трудоемкой задачей, поэтому вполне понятным является наличие на сегодняшний день всего трех хорошо отлаженных наборов функций уровней ядр**а** (“движк**а**”) броузеров.

Современные RAD-системы разработки приложений Delphi / C++Builder штатно включают высокоуровневые компоненты, инкапсулирующие значительные участки (подобного вышеприведенному) кода сетевых приложений. Например, в Delphi версии 5 включен компонент типа TWebBrowser (TCppWebBrowser для C++Builder), позволяющий создать функционирующий (полностью подобный MS Explorer’у благодаря вызовам тех же процедур из предоставленных MS объектов ActiveX) броузер путем введения всего одной-двух строк исходного кода, однако все (крайне ценные для реальных программистов) тонкости HTTP-программирования при этом утрачиваются. Интересующихся программированием броузеров на уровне гнезд (сокетов) отсылаем к перечисленным в списке литературы работам, ниже будет рассмотрена технология создания броузера на “среднем” уровне – уровне компонента NMHTTP фирмы NetMasters L.L.C, www.netmastersllc.com (данный компонент инкапсулирует технологию сокетов и благодаря этому обеспечивает гибкое управление соединением, что позволяет расширять функциональность готового приложения по желанию пользователя). При этом возможности NMHTTP велики (например, с его помощью можно организовывать обновление контента сервера).

Важные свойства, методы и события компонента NMHTTP приведены в табл.3 ÷ 5.

Таблица 3.— Свойства компонента NMHTTP.

Свойство и его тип	*Описание*
Body (string)	Заключает в себе тело документа, с которым оперирует NMHTTP, функционирование определяется значением свойства InputFileMode. При InputFileMode=True, в свойстве Body находит имя файла, в который будет помешено тело документа; в прот**и**вном случае в это поле помещается сам документ.
BytesRecvd (longint,	Количество принятых от удаленного сервера байтов на данный

read only)	момент.
BytesSent (longint, read only)	Количество байтов, посланных удаленному серверу на данный момент.
BytesTotal (longint, read only)	Общее количество байтов в данной HTTP-сессии (может использоваться как при получении документа, так и при его отправке).
CookieIn (string, read only)	Содержит Cookie, полученные от удаленного сервера.
Header (string)	Содержит заголовок документа, с которым работает NMHTTP.
HeaderInfo (THeaderinfo)	Содержит информацию об *отсылаемом* документе. В его состав входят следующие пункты: - Headerinfo.Cookie (string). Содержит в себе текст Cookie, отсылаемого вместе с документом. - Headerinfo.LocalMailAddress (string). Содержит E-Mail адрес отравителя. - Headerlnfo.LocalProgram (string). В этом свойстве содержится имя программы, формирующей ответ удаленному HTTP-серверу - Headerinfo.Password (string). Содержит пароль для аутентификации пользователя на удаленном сервере, если это требуется. - Headerinfo.UserID (string). Содержит идентификатор пользователя для аутентификации его на удаленном сервере, если это необходимо.
Host (string)	Имя сервера, с которым будет произведен сеанс связи во время следующей HTTP-транзакции (используется редко).
InputFileMode (boolean)	Определяет порядок использования свойств Body и Header. При InputFileName=False Body и Header получат содержимое тела и заголовка документа соответственно, в противном случае в них должны быть указаны имена html-файлов, в которые будут помещены тело и заголовок документа.
LastErrorNo (integer, read only)	Номер последней ошибки, произошедшей во время HTTP-сеанса.
LocalIP (string)	Содержит IP-адрес пользователя (если компьютер пользователя имеет более одного IP-адреса, указывается первый).
OutputFileMode (boolean)	Играет ту же роль, что и свойство InputFileMode, но только уже для посылаемых данных.
Port (integer)	Номер порта, на котором функционирует удаленный сервер, с которым устанавливается соединение (по умолчанию порт 80).
Proxy (string)	IP-адрес используемого proxy-сервера (если proxy-сервер не используется, должен быть установлен пробел).
FroxyPort (integer)	Номер порта, на котором функционирует proxy-сервер.
RemoteIP (string)	IP-адрес удаленного сервера, с которым будет произведено соединение.
ReplyNumber (smallint, read only)	Содержит стандартный номер ответа удаленного сервера.
ReportLevel (integer)	Указывает степень детализации информации о статусе при инициации события OnStatus.
SendHeader (string)	Заголовок документа, который посылается удаленному серверу.
Status (string, read	Сообщение о статусе, сгенерированное во время последней отра-

only)	ботки события OnStatus.
TimeOut (integer)	Время (в миллисекундах) ожидания ответа от удаленного сервера (после его истечения возбуждает исключение и прерывает текущую операцию, при равенстве нулю исключение не возбуждается).
TransactionReply (string, read only)	Результат последней команды, посланной на удаленный сервер.
WSAInfo (TStringList, read only)	Информация о текущей версии Winsock.

Таблица 4.— Методы компонента NMHTTP.

Прототип	*Описание*
Procedure Abort	Прерывает текущую транзакцию HTTP.
Procedure Delete (URL: string)	Физически удаляет документы, расположенные на сервере, адрес которого указан в параметре URL.
Procedure Get (URL: string)	Запрашивает и получает документ, находящийся по указанному в параметре URL адресу. Тело и заголовок документа размещаются в свойствах Body и Header соответственно (с учетом значения свойства InputFiieOpen).
Procedure Head (URL: string)	Извлекает заголовок документа, находящегося по указанному в параметре метода адресу (заголовок помещается в свойство Header).
Procedure Options (URL: string)	Получает список опций для указанного URL и помещает его в свойство Body (с учетом состояния свойства InputFileMode).
Procedure Post(URL, PostData: string)	Предназначен для посылки данных на сервер, чей адрес указан в параметре URL. В зависимости от значения свойства OutPutFileString в параметре PostData указано либо имя файла, в котором эти данные находятся, либо сами отсылаемые данные.
Procedure Trace(URL, TraceData: string)	Метод Trace запрашивает URL, указанный в параметрах метода для отображения трассировочных данных, которые помещаются в параметр TraceData в соответствии со значением свойства OutputFileMode (большинство серверов не поддерживают эту возможность).

Таблица 5.— События, генерируемые компонентом NMHTTP.

Наименование	*Описание*
OnAboutToSend	Возникает перед посылкой заголовка удаленному серверу.
OnConnect	Генерируется в случае удачного соединения с сервером.
OnConnectionFailed	Генерируется в случае неудачной попытки соединения с удаленным сервером.
OnDisconnect	Возникает при штатном разрыве связи с сервером.
OnFailure	Возникает в случае сбоя во время HTTP-сессии (обратите внимание на разницу между этим событием и OnConnectionFailed).
OnHostResolved	Возникает при перемене представления удаленного сервера (на-

	пример, при смене его имени на IP-адрес).
OnInvalidHost	Генерируется в случае, если в свойстве Host указано неверное или несуществующее имя удаленного сервера.
OnPacketRecvd	Генерируется каждый раз, когда отдельный пакет принимается от удаленного сервера в рамках HTTP-сессии (событие удобно при организации индикатора прогресса приема).
OnPacketsent	Возникает каждый раз, когда пакет данных пересылается удаленному серверу.
OnStatus	Генерируется каждый раз, когда удаленный сервер очередное сообщение об изменении статуса.

При создании простейшего WEB-броузера обрабатываются (в качестве примера) только события OnPacketRecvd и OnStatus, используются методы Get и Stop:

```
procedure TForm1.NMHTTP1PacketRecvd(Sender: TObject);
{ вызывается при наступлении события OnPacketRecvd объекта NMHTTP1 }
begin
 StatusBarInfo.SimpleText:=IntToStr(NMHTTP1.BytesRecvd)+
                           ' байт из общего числа ' +IntToStr(NMHTTP1.BytesTotal)+
                           ' получено';
end; { конец NMHTTP1PacketRecvd }

procedure TForm1.NMHTTP1Status(Sender: TComponent; Status: String);
{ вызывается при наступлении события OnStatus объекта NMHTTP1 }
begin
 if NMHTTP1.ReplyNumber = 404 then { совсем плохо... }
  StatusBarInfo.SimpleText:='Запрошенный ресурс не найден';
end; { конец NMHTTP1Status }

procedure TForm1.ButtonConnectClick(Sender: TObject);
{ нажатие кнопки Connect }
begin
 if EditURL.Text = '' then { если пусто }
 begin
  ShowMessage('Поле адреса WEB-сервера не может быть пустым...');
  exit;
 end;

 MemoHeader.Clear; { очистка полей вывода }
 MemoBody.Clear;
 EditCookie.Clear;

 NMHTTP1.InputFileMode := FALSE;
 NMHTTP1.OutputFileMode := FALSE;
 NMHTTP1.ReportLevel := Status_Basic;

{ настройка броузера }
```

```
 NMHTTP1.HeaderInfo.UserId:=EditUser.Text; { идентификатор пользователя }
 NMHTTP1.HeaderInfo.Password:=EditPassword.Text; { пароль }
 if EditPort.Text <> '' then { если непусто... }
  NMHTTP1.Port:=StrToInt(EditPort.Text); { номер порта сервера }

 NMHTTP1.Proxy := '';
 NMHTTP1.ProxyPort := 0;
 if CheckBox1.Checked then
 begin
  NMHTTP1.Proxy := EditProxy.Text;
  NMHTTP1.ProxyPort := StrToInt(EditProxyPort.Text);
 end;

 StatusBArInfo.SimpleText:='Выполнение запроса';
 NMHTTP1.Get(EditURL.Text); { имя запрашиваемого файла взять из строки ввода
}

 StatusBArInfo.SimpleText:='Запрос выполнен';

 LabelIP.Caption := NMHTTP1.LocalIP + ' -> ' + NMHTTP1.RemoteIP; { показать IP-
адрес сервера }

 StatusBArInfo.SimpleText:='Показываю заголовок сообщения...';
 MemoHeader.Text:=NMHTTP1.Header; { показать заголовок ответа от сервера }

 StatusBArInfo.SimpleText:='Показываю тело сообщения...';
{ запрошенный файл получен в NMHTTP1.Body. Именно теперь следует опи-
  сать процедуру его анализа, запроса составляющих (изображений etc) и
  вЫдачи в графическом виде - как в привычных броузерах
= Вместо этого просто передаем принятый HTML-текст в поле MemoHTML... = }
MemoBody.Text:=NMHTTP1.Body; { показать контент ответа от сервера }

 StatusBArInfo.SimpleText:='Показываю Cookies...';
 EditCookie.Text:=NMHTTP1.CookieIn; { показать Cookie }

 StatusBArInfo.SimpleText:='Сеанс закончен. С сервера принято '
            + IntToStr(NMHTTP1.BytesRecvd) + ' байтов';

end; { конец программы ButtonConnectClick приема и показа данных }

procedure TForm1.ButtonSaveClick(Sender: TObject);
{ нажатие кнопки Save }
begin
 if SaveDialog1.Execute then
  MemoHTML.Lines.SaveToFile(SaveDialog1.FileName);
end; { конец ButtonSaveClick }

procedure TForm1.ButtonStopClick(Sender: TObject);
{ нажатие кнопки Stop }
```

```
begin
 NMHTTP1.Abort; { экстренно прекратить связь }
end; { конец ButtonStopClick }

procedure TForm1.ButtonBrowseClick(Sender: TObject);
{ нажатие кнопки Browse }
var
 FileName: string;
begin
 if MemoHTML.Lines.Count > 0 then { если что-то есть в MemoHTML }
 begin
  FileName:='$temp$.htm'; { имя временного файла }
  MemoHTML.Lines.SaveToFile(FileName); { сохранить во временный файл }
  ShellExecute(0, NIL, { по умолчанию = open } { вызвать штатный броузер }
                  PChar(FileName), NIL, NIL,
                  SW_SHOWNORMAL);
 end;
end; { конец ButtonBrowseClick }
```

Программа выводит полученные от сервера данные в текстовое окно, не делая попыток интерпретировать HTML-команды (эту, в общем-то несложную, но трудоемкую процедуру может организовать программист дополнительно). Кнопки Сохранить и Отобразить (инициируют выполнение процедур ButtonSaveClick и ButtonBrowseClick соответственно) служат для сохранения принятого текста в файл и интерпретации его штатным системным броузером.

Необходимое оборудование – IBM PC-совместимая ЭВМ, предустановленная ОС Windows, пакет Delphi версии выше 4.

Порядок проведения работы. Студент знакомится с правилами построения клиентских приложений на основе высокоуровневых компонент, собирает клиентское приложение Browser.exe, проверяет функционирование полученной системы.

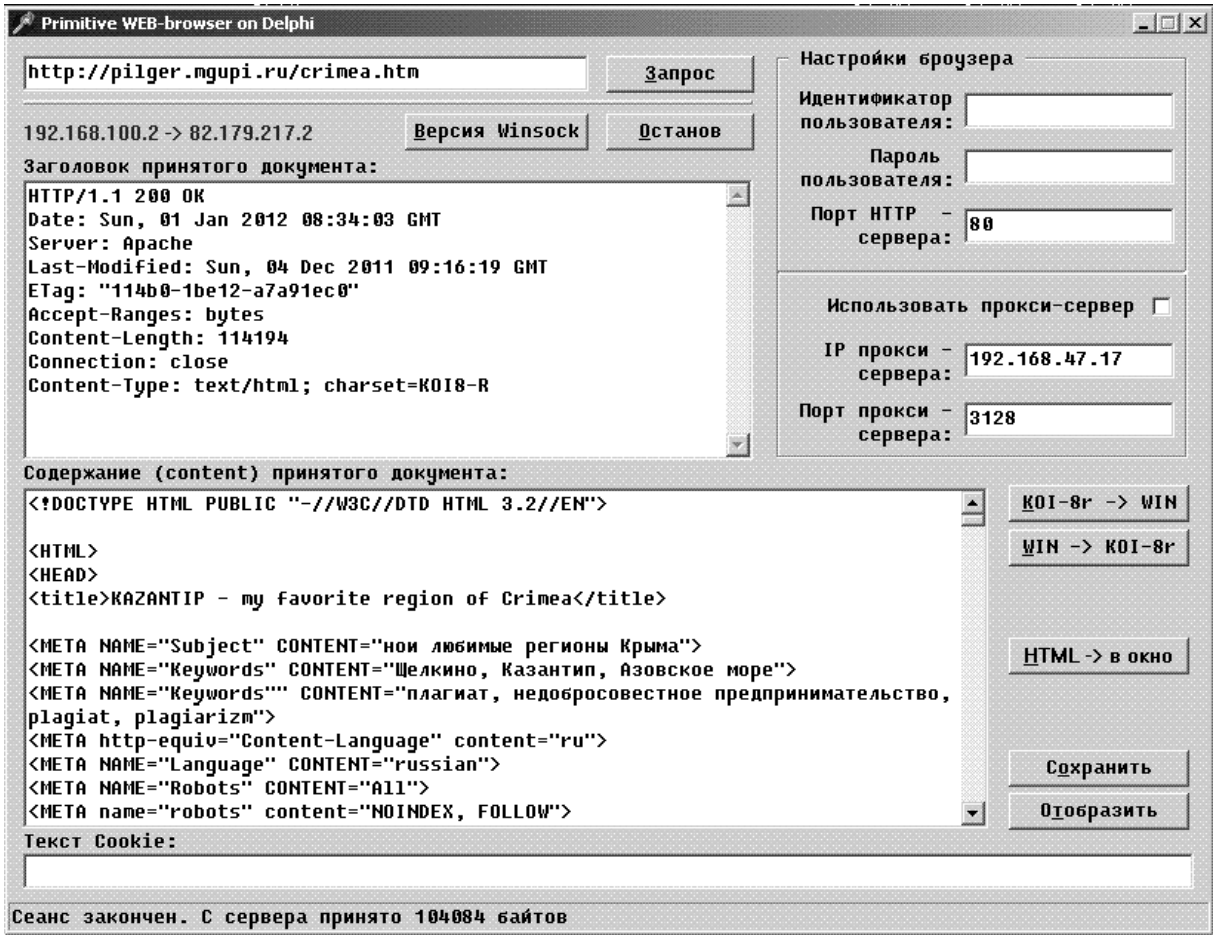

Рисунок 14.— Окно приложения Browser

Компиляция приложения Browser проводится стандартными методами (используются файлы Browser.dpr, Browser.dof, Browse.pas, Browse.dfm). Проверка функциональности может быть проведена на локальной машине в offline-режиме (при использовании WEB-сервера Apache, его установка и настройка описаны ранее) или в Сети (может потребоваться дополнение файла HOSTS).

Как видно из рис.14, принимаемые от сервера данные выдаются в виде сплошного текста; синтаксический анализ и графическая интерпретация HTML-текста (выделение тегов и гиперссылок, иллюстраций и др. объектов, корректное размещением их на странице в графическом режиме – может представлять интерес компонент типа THTML) может явиться (громоздким) заданием для самостоятельной работы студенту. Кнопка “Сохранить” служит для сохранения HTML-текста в файл, с помощью кнопки “Отобразить” принятое сообщение (исключая заголовок) передается штатному броузеру для отображения. С целью установки режима работы без прокси-сервера следует присвоить нулевое значение номеру порта прокси (NMHTTP1.ProxyPort:=0;).

В общем случае для выгрузки заданного файла из Сети с помощью компонента NMHTTP достаточно выполнить следующий участок кода:

```
NMHTTP1.InputFileMode:=True; { установить режим работы с файлом }
NMHTTP1.Get('http://pilger.mgapi.edu/crimea.htm'); { имя ресурса для выгрузки }
NMHTTP1.Body:='d:\crim.htm'; { имя файла сохранения выгруженного ресурса }
```

По заданию преподавателя приложение Browser может быть доработано (домашнее задание повышенной сложности), например:

- Переработан (с целью повышения удобства пользователя) интерфейс WEB-броузера.
- Введен максимально подробный анализ ошибок сеанса связи с сервером.
- Анализ внутренних гиперссылок и реализация многопоточности для из (одновременной) выгрузки с сервера объектов HTML-страницы.

Для более глубоко заинтересованных технологией применения высокоуровневых объектов Delphi / C++Builder рекомендуется рассмотреть создание клиента FTP (с использованием компонента NMFTP, страница InterNet палитры компонентов) и Finger-клиента (компонент NMFinger), реализацию электронной почты (компоненты NMPOP3 и NMSMTP) и др. (дополнительные данные могут быть получены из [2,4,5]).

Оформление отчета по работе. В отчете приводятся список необходимых для реализации заданной функциональности программы высокоуровневых объектов (компонентов) Delphi / C++Builder и информация по их настройке и связыванию в действующем приложении. В отчете должна присутствовать функциональная часть содержимого файла HOSTS и несколько $(3 \div 5)$ заголовков ответов WEB-сервера броузеру (с указанием местоположения принятых объектов в окне последнего при визуализации принятой информации).

Вопросы для самопроверки.

1. Назвать основные функции ПО клиентской стороны сети InterNet.
2. Какие наборы функций ядра (‘движков’) броузеров известны на сегодняшний день?
3. В чем заключаются преимущества и недостатки использования высокоуровневых компонент при создании InterNet-приложений?

Список литературы

1. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы (учебник для ВУЗ'ов). —СПб.: Питер, 2011. —944 с.
2. Э.Таненбаум, Д.Уэзеролл. Компьютерные сети. —СПб.: Питер, 2012. —960 с.
3. Исаченко О.В.. Программное обеспечение компьютерных сетей. —М.: Инфра-М, 2012. —120 с.
4. Смелянский Р.Л. Компьютерные сети (в 2 томах). —М.: Академия, 2011 (том 1: Системы передачи данных, —304 с.; том 2: Сети ЭВМ, —240 с.).
5. Баканов В.М. Сетевые технологии: учебное пособие (курс лекций). —М.: МГУПИ, 2008. –105 с.

Учебное издание.

Баканов Валерий Михайлович

Сетевые технологии

Учебно-методическое пособие по
выполнению лабораторных работ

Подписано в печать 12.03.2013 г.
Формат 60 × 84 1/16.
Объем 4,0 п.л. Тираж 100 экз. Заказ 36.

Отпечатано в типографии Московского государственного
университета приборостроения и информатики.
107846, Москва, ул.Стромынка,20.