Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра ЭМИС

Касимов В.З.

ЯЗЫКОВЫЕ СРЕДСТВА СОЗДАНИЯ ГИПЕРДОКУМЕНТОВ

Лабораторные работы

Учебно-методическое пособие
для студентов направления 230200
«Информационные системы»

2011

Касимов В.З.
Языковые средства создания гипердокументов. Лабораторные работы.
Учебно-методическое пособие. – Томск: ТУСУР, 2011. – 14 с.

Учебно-методическое пособие посвящено основам практического применения языка html как языка гипертекстовой разметки и работе с интерфейсом сетевого программного обеспечения Winsock.

Пособие рассчитано на студентов вузов.

Введение

Этот курс лабораторных работ предназначен для практического изучения основ применения языка html как языка гипертекстовой разметки и работе с интерфейсом сетевого программного обеспечения Winsock. В методических указаниях основное внимание уделяется принципиальным моментам, которые необходимы для их успешного выполнения.

Предполагается, что обучающийся владеет навыками программирования на языке С.

Лабораторная работа №1

Создание Web-страницы с текстом

Поскольку язык HTML является не языком программирования, а языком разметки гипертекстовых документов, то для написания простейшей Web-страницы достаточно любого текстового редактора, например, стандартного приложения Windows Блокнот.

Введем следующий текст в редактор, сохраним его в файле с расширением html и откроем в браузере:

```
<HTML>
<HEAD>
<TITLE>Заголовок документа</TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Это будет являться заготовкой документа. В данной лабораторной работе продемонстрируем действия основных тегов HTML, связанных с заголовками различного уровня, форматированием текста, организацией списков и таблиц.

Содержимое Web-страницы вводится после тега BODY.

Теги для заголовков различного уровня:

```
<H1>Заголовок первого уровня</H1>
<H2>Заголовок второго уровня</H2>
<H3>Заголовок третьего уровня</H3>
<H4>Заголовок четвертого уровня</H4>
```

```
<H5>Заголовок пятого уровня</H5>
<H6>Заголовок шестого уровня</H6>
```

Теги форматирования шрифта:

```
<B>Жирный</B>
<I>Курсив</I>
<U>Подчеркнутый</U>
<B><I><U>Жирный подчеркнутый курсив</B></I></U>
<TT>Равноширинный</TT>
```

Теги выделения:

```
<EM>Выделение</EM>
<STRONG>Усиленное выделение</STRONG>
```

Теги нумерованного списка:

```
<OL>
<LI>Первый элемент списка</LI>
<LI>Второй элемент списка</LI>
</OL>
```

Теги ненумерованного списка:

```
<UL>
<LI>Первый элемент списка</LI>
<LI>Второй элемент списка</LI>
</UL>
```

Теги списка определений:

```
<DL>
<DT>Определение 1</DT>
<DD>Пояснение к определению 1</DD>
</DL>
```

Цветовая схема страницы вводится внутри тега BODY заданием атрибутов в виде ATTR=«#RRGGBB», где в качестве ATTR может быть цвет фона BGCOLOR, цвет текста TEXT, цвет текста ссылки LINK, цвет текста активной ссылки ALINK, цвет текста просмотренной ссылки VLINK.

Задание

Создать страницу о себе. Продемонстрировать все возможности форматирования текста, использование списков, задания цвета, работы с таблицами.

Лабораторная работа №2

Создание простейшего приложения, использующего Winsock. Разрешение имен.

Winsock – это часть API Windows для разработки сетевых приложений Windows. Он представляет собою интерфейс между приложением и транспортным протоколом, выполняющим передачу данных. Для создания простейшего приложения, использующего Winsock, нужно создать проект типа Win32, указав шаблон проекта *Консольное приложение Win32*, в дополнительных параметрах необходимо выбрать *Предварительно скомпилированный заголовок* и установить флажок *MFC*. В файл stdafx.h проекта добавить строку

```
#include <winsock2.h>
```

а на вкладке свойств проекта в разделе *Компоновщик* указать дополнительную зависимость Ws2_32.lib.

Инициализация Winsock осуществляется функцией WSAStartup, которая объявлена как:

```
int WSAStartup(  WORD wVersionRequested, LPWSADATA lpWSAData  );
```

Первый параметр – это требуемая версия Winsock (младший байт определяет номер основной версии, старший байт – дополнительный номер версии. В случае успеха функция возвращает 0. Второй параметр – это указатель на структуру WSADATA, в которой возвращаются параметры инициализации.

```
typedef struct WSAData {
        WORD wVersion;
        WORD wHighVersion;
        char szDescription[WSADESCRIPTION_LEN+1];
        char szSystemStatus[WSASYS_STATUS_LEN+1];
        unsigned short iMaxSockets;
        unsigned short iMaxUdpDg;
        char FAR * lpVendorInfo;
} WSADATA, FAR * LPWSADATA;
```

Вообще говоря, если требуется точное соответствие номера версии, необходимо даже в случае возврата 0 сравнение wVersionRequested с полем wVersion структуры WSAData. Если это не так, то нужно обратиться к функции WSACleanup, которая завершает использование данного API. В случае соответствия версии можно продолжать работу с Winsock. В конце работы с Winsock также необходимо вызвать WSACleanup, чтобы уменьшить на 1 счетчик использования Winsock, поскольку Winsock реализована как DLL. Данная функция при удачном выполнении вернет 0.

Задание

Написать функции

```
void Startup(WORD wVersion, CString *pS)
void Cleanup(CString *pS),
```

которые проводят соответственно инициализацию Winsock и зачистку использования DLL. Если указатель pS отличен от 0, то по этому адресу записывается диагностический текст. В первой функции инициализация должна проводиться точно по версии wVersion. Проверить выполнение данных функций, используя последовательный вызов с печатью диагностических сообщений.

Порядок байт

Семейство протоколов Интернет используется в разнородной сети, в которой могут присутствовать вычислительные системы с различной адресации многобайтовых данных. В связи с этим различают сетевой порядок байтов (от старшего к младшему) и системный (от младшего к старшему). По сети данные передаются в сетевом порядке. Для преобразования данных существуют 4 функции:

u_long htonl(u_long hostlong)

u_long ntohl(u_long netlong)

u_short htons(u_short hostshort)

u_short ntohs(u_short netshort).

При подготовке многобайтовых данных для передачи по сети их нужно

перевести в сетевой порядок байтов.

Разрешение имен

Для подключения к узлу по IP-протоколу требуется знать IP-адрес узла, хотя более удобно знание доменных имен. Определение IP-адреса по доменному имени узла называется разрешением имени. Здесь следует иметь в виду, что для одного доменного имени может существовать несколько IP-адресов. Операция разрешения имени осуществляется на основе функции gethostbyname, знания структуры hostent.

Задание

Написать функции

```
BOOL getIPForName(CString& sName, CStringList& lIP)
BOOL getIPForName(CString& sName, CList<unsigned long>& lIP)
```

Обе функции принимают в качестве входного параметр sName. Это может быть как правильное текстовое представление IP-адреса, так и доменное имя. Второй параметр является выходным. В первом случае – это список строк, представляющий собой множество правильных текстовых представлений IP-адреса, во втором – список IP-адресов в виде unsigned long в сетевом порядке байтов. Возвращаемое значение TRUE означает успешное выполнение функции. Для правильной реализации задания требуется также знание функции inet_addr.

Обратная операция – получение по IP-адресу доменного имени, производится с помощью метода gethostbyaddr.

Задание

Написать функцию

```
BOOL getNameForIP(CString& sIP, CString& sName)
```

Здесь sIP – текстовое представление IP-адреса (входной параметр), sName – доменное имя (выходной параметр).

Лабораторная работа №3

Работа с сокетами. Клиенты и серверы. Прием и передача данных в блокирующем режиме.

Основным средством в Winsock для приема и передачи информации является сокет (socket). Поскольку мы программируем для сети Интернет, то нам доступны 2 сетевых протокола: UDP/IP и TCP/IP. Первый протокол – без установления соединения, но сохраняющий границы сообщений. Второй – с установлением соединения, но не сохраняющий границы сообщений и рассматривающий принимаемую информацию как поток. Первый протокол имеет меньшие накладные расходы, но недостаточно надежен. Второй – надежен, гарантирует доставку пакетов информации и формирование из них сообщений, но имеет большие накладные расходы. Мы будем рассматривать протокол TCP/IP. Для него сокет создается с помощью вызова:

```
SOCKET s = socket(AF_INET, SOCK_STREAM, 0)
```

Здесь `SOCKET` – тип данных, `AF_INET` – параметр, задающий семейство протоколов (Интернет), `SOCK_STREAM` – тип сокета (для UDP/IP – `SOCK_DGRAM`). При ошибке функция возвращает INVALID_SOCKET (-1). В этом случае можно получить расширенную информацию об ошибке вызовом WSAGetLastError().

После создания сокета последовательность действий отличается в зависимости от того, какую роль выполняет разрабатываемое приложение: клиента или сервера.

Последовательность команд клиента

Для соединения с сервером после создания сокета требуется провести 2 действия: разрешить IP-адрес сервера (смотри лабораторную работу №2) и произвести соединение с сервером.

Для соединения с сервером требуется заполнить структуру sockaddr_in:

```
sockaddr_in addr;
ZeroMemory( &addr, sizeof(addr));
// тип адреса (TCP/IP)
addr.sin_family = AF_INET; // семейство протокола Интернет
addr.sin_addr.S_un.S_addr = inet_addr(«127.0.0.1»);
// для заполнения этого поля можно также использовать вызов
```

```
// getIPForName(CString& sName, CList<unsigned long>& lIP)
s_addr.sin_port = htons(5050); // номер порта сервера в сетевом порядке байт
```

и выполнить соединение вызовом

```
if( connect(s,(sockaddr*)&addr, sizeof(addr)) ) == SOCKET_ERROR ) {
  // Расширенный код ошибки можем получить вызовом
  WSAGetLastError();
}
```

Если ошибки не произошло, то после соединения проводится обмен данными через сокет s.

Последовательность команд на стороне сервера

Сервер – это вычислительный процесс, ожидающий подсоединения клиента и обслуживающий его запросы. После создания сокета он связывается с сетевым интерфейсом, на котором производится прослушивание возможных соединений со стороны клиентов, вызовом функции bind:

```
int bind (
 SOCKET s, // сокет прослушивания
 const struct sockaddr FAR* name,
 int namelen
);
```

Здесь поля структуры name, которая на самом деле есть структура типа sockaddr_in, заполняется также, как и для клиента. Только в качестве поля `sin_addr.S_un.S_addr` задается либо конкретный IP-адрес сервера, на котором ведется прослушивание, либо константа INADDR_ANY, означающий «любой» адрес. В качестве номера порта задается номер порта сервера.

Перевод сокета в режим ожидания входящих соединений производится вызовом listen(s, int l), l – максимальная длина очереди сообщений, ожидающий обработки.

Возвращаемое функцией accept значение позволяет получить сокет соединения с клиентом:

```
SOCKET accept(s, SOCKADDR *addr, int *addrLen)
```

Здесь информация, получаемая по адресу addr, будет содержать сведения о клиенте, в частности, его IP-адрес. Сервер, обслуживающий более одного клиента, должен после возврата из accept создать рабочий поток для облуживания клиента и снова вызвать accept для продолжения прослушивания. Таким образом, на стороне сервера первичный сокет всегда находится на про-

слушивании соединения клиентов, а возвращаемый функцией accept сокет предназначен непосредственно для обмена данными с клиентом.

Передача и прием данных

Для передачи данных по сокету используется функция send. Аналогично прием данных осуществляется функцией recv. Эти функции при успешном выполнении возвращают общее число переданных (принятых) байт, в противном случае – SOCKET_ERROR (-1). Наиболее распространенными расширенными кодами ошибки, который возвращается WSAGetLastError(), являются:

- WSAECONNABORTED при разрыве виртуального соединения из-за ошибки протокола или истечения времени ожидания;

- WSAECONNABORTED при закрытии соединения партнерским узлом;

- WSAEWOULDBLOCK при использовании асинхронных режимов работы сокета (функция не может быть выполнена в настоящий момент).

Функция send определяется следующим образом:

```
int send(SOCKET s, char *buf, int len, int flags)
```

Здесь buf – символьный буфер, len – длина блока передаваемых данных, значение flags может быть равно 0.

Функция recv определена точно так же, как send.

Прежде всего заметим, что при достаточно большом объеме принимаемой (передаваемой) информации, проблематично произвести обмен за один вызов функции. В этом случае требуется неоднократный вызов со сдвигом адреса информации с учетом уже принятой (переданной).

Далее, при передаче данных известно, сколько данных следует передавать, а при приеме в общем случае – нет, поскольку сокет – потоковый без сохранения границ сообщения. В этом случае существует только один выход: решение вопроса о формате данных в рамках протокола уровня приложения.

По завершении обмена необходимо закрыть сокет вызовом функции closesocket(SOCKET s), однако перед ее вызовом, чтобы избежать потери данных, следует вызвать функция shutdown:

```
int shutdown(SOCKET s, int how).
```

Параметр how (SD_RECEIVE, SD_SEND, SD_BOTH) определяет, какие действия по сокету в дальнейшем запрещены.

Задание

Написать функции

```
int Send(SOCKET s, char *buf, int len, int mode)
int Recv(SOCKET s, char *buf, int len, int mode)
```

для обмена данными через сокет путем неоднократного вызова send и recv. Параметр len определяет общее количество передаваемых байт (для Send) или размер буфера (для Recv). Параметр mode может принимать 3 значения: 4, 2 или 0. Он означает количество байтов, в которых в начале сообщения в формате целого числа (в сетевом порядке) кодируется число передаваемых далее байтов сообщения.

На основе этих функций создать 2 приложения: клиент и эхо-сервер. Сервер после запуска становится в режим прослушивания и после установления клиентом соединения считывает данные и возвращает их обратно клиенту. Клиент после установления соединения передает серверу произвольные данные и считывает полученные от сервера. Проэкспериментировать с работой клиента и сервера на реальной сети компьютерного класса, передавая данные достаточно большого объема, установив, какова характерная величина порции данных, передаваемых за 1 вызов send.

Лабораторная работа №4
Режимы работы сокета. Модели ввода-вывода. Прием и передача данных в неблокирующем режиме.

Winsock поддерживает 2 режима работы сокета: блокирующий и неблокирующий. По умолчанию сокет создается в блокирующем режиме. В этом режиме функции ввода-вывода ожидают завершения операции, в неблокирующем завершаются немедленно, как правило, с расширенным кодом

ошибки WSAEWOULDBLOCK. При использовании блокирующего режима работы сокета и однопотокового приложения практически невозможно одновременно производить прием и передачу данных, что сильно сужает функциональность сетевого приложения. Разделение приложения на считывающий и вычислительный потоки позволяет производить одновременный прием и передачу данных. При этом должны использоваться объекты синхронизации. Однако такой подход ограничен в смысле масштабирования, если требуется работа с несколькими клиентами.

Для асинхронной работы с несколькими сокетами Winsock поддерживает несколько моделей ввода-вывода, которые работают в неблокирующих режимах сокета. Рассмотрим кратко две таких модели ввода-вывода.

Модель ввода-вывода WSAAsyncSelect

Данная модель ввода-вывода ориентирована на приложения Windows с графическим интерейсом. Как известно, такие приложения имеют хотя бы один объект ядра типа окно. Сетевые события посылают уведомления в виде оконных сообщений. Уведомления активизируются путем вызова функции:

```
int WSAAsyncSelect(  SOCKET s,
                     HWND hWnd,
                     unsigned int wMsg,
                     long lEvent)
```

После вызова данная функция переводит сокет s в неблокирующий режим. Уведомления посылаются окну с дескриптором hWnd в виде сообщения с кодом wMsg. Комбинация сетевых событий, по которым производится уведомление, задается в параметре lEvent посредством битовой маски, составленной из констант, наиболее упетребимыми из которых являются: FD_READ – готовность сокета к чтению, FD_WRITE – готовность сокета к записи, FD_ACCEPT – имеется входящее соединение, FD_CONNECT – произошло соединение с сервером, FD_CLOSE – сокет закрыт партнерским узлом. Очевидно, что в зависимости от типа приложения (клиент или сервер), фазы выполнения приложения требуются разные события, что достигается

неоднократными вызовами WSAAsyncSelect.

В общем случае возможно регистрировать уведомления как на разные окна, так и на разные коды сообщений. При посуплении уведомления функция окна получает сообщение, из параметров которого можно извлечь дескриптор сокета, код ошибки и тип сетевого события, инициировавшего сообщение.

Модель ввода-вывода select

Модель ввода-вывода названа по имени соответствующей функции:

```
int select(     int nfds,             // несущественно
                fd_set *read,         // набор событий чтения
                fd_set *write,        // набор событий записи
                fd_set *except,       // набор событий неудачного
                                      // установления соединения
                timeval *timeout)     // структура описания тайм-аута
```

Функция блокирует операции ввода-вывода, пока не произойдут какие-либо изменения в одном из заданных параметров типа fd_set, сигнализирующие о событиях. При этом возвращаемое функцией значение равно количеству изменений. После этого остальные элементы fd_set, по отношению к которым сетевых событий не произошло, сбрасываются. После возврата из функции несброшенные события могут быть проанализированы и произведены соответствующие вызовы функций в ответ на сетевое событие.

Для работы с fd_set существуют соответствующие макросы, позволяющие очистить набор, удалить сокет из набора, проверить, имеется ли сокет в наборе, добавить сокет в набор.

Приложение, использующее данный набор ввода-вывода, представляет собой бесконечный цикл, в начале которого формируются нужные наборы в зависимости от фазы работы с сокетом и производится вызов функции select. После возврата анализируются наборы событий и производится реакция на изменения, после чего производится переход к началу цикла.

Задание

Используя модель ввода-вывода select, написать простой http-сервер, ко-

торый работает по порту 80 и сразу после подключения к нему клиента посылает ему содержимое html-файла из лабораторной работы №1. Проверить работу сервера, используя в качестве клиента Internet Explorer.

Рекомендуемая литература

1. Мэйерс К.Р. HTML: Пер. с англ. / К. Р. Мэйерс ; Под ред.Ф.Романо. - М. : МГУП, 2005. - 62 с.

2. Пауэлл Т.А. Web - дизайн : Наиболее полное руководство в подлиннике: Пер. с англ. / Т. А. Пауэлл. - СПб. : БХВ-Петербург, 2002. - 996 с. - ISBN 5-94157-102-X : 203.15.

3. Петюшкин, А.В. HTML в Web-дизайне / А. В. Петюшкин. - СПб. : БХВ-Петербург, 2004. - 400 с.

4. Джонс Э., Оланд Д. Программирование в сетях Microsoft Windows – СПб.: Питер, 2002. - 608с.