МГАПИ

МОСКОВСКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ
ПРИБОРОСТРОЕНИЯ И ИНФОРМАТИКИ

Кафедра ‘Персональные компьютеры и сети’

В.М.Баканов

Программное обеспечение компьютерных сетей и информационных систем

(конспект лекций)

Москва
2003

АННОТАЦИЯ

Данное пособие предназначено для студентов IV-V курсов, обучающихся по курсу ‘Программное обеспечение компьютерных сетей и информационных систем’ или создающих программное обеспечение (ПО) указанного класса в среде различных операционных систем (ОС), может быть применено в качестве курса лекций и для самостоятельной работы (содержит ссылки на дополнительные источники информации, в т.ч. InterNet). Хотя в данной работе рассматриваются в основном компьютерные сети на основе IBM PC-совместимых персональных ЭВМ и ОС WINDOWS (особенно WINDOWS с ядром NT), основные положения верны и для ЭВМ общего класса.

В пособии изложены основы функционирования штатного ПО поддержки сетевого взаимодействия ЭВМ (рассматриваются механизмы NetBIOS, RPC, Windows Sockets и др.), приведены данные об современных основных сетевых протоколах, содержится практический материал для разработчиков оригинального сетевого ПО, дана вводная информация о кластерных вычислительных системах, серьезное внимание уделено глобальной сети InterNet.

Практические примеры рассчитаны на использование интегрированных сред Visual C++ корпорации Microsoft и C++Builder фирмы Borland Int. (сейчас Inprise Corp.) и выполняются при поддержке 32-х битовых ОС семейства WINDOWS.

Требуемое аппаратное обеспечение для проведения работ - ПЭВМ класса IBM PC с процессором не хуже i586 с объемом оперативной памяти 64-128 Mb, твердым диском не менее 4-6 Gb, дисплеем класса не хуже VGA и соответствующими сетевыми средствами - стандартным сетевым контроллером (для непосредственного подключения к локальной сети) и/или модемом (для работы в сети InterNet через телефонный канал). Последняя версия данного методического пособия может быть свободно выгружена в виде файла http://pilger.mgapi.edu/methods/nets.zip.

Автор: доцент, д.т.н. Баканов В.М.
Рецензент: профессор Рощин А.В.
Научный редактор: профессор, д.т.н. Михайлов Б.М.
Работа рассмотрена и одобрена на заседании кафедры
ИТ-4 МГАПИ _____ 2003 года.

Заведующий кафедрой ИТ-4 профессор, д.т.н. Б.М.Михайлов

Предыдущее издание:

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

История объединения компьютеров между собой почти столь же стара, как и история создания собственно компьютеров [1]. Еще на заре компьютерной эры пользователи ЭВМ поняли, что намного проще обмениваться данными между машинами по кабелю (использовались поддерживаемые со времен Norton Commander'а для DOS параллельный или последовательный интерфейсы), чем применять перенос данных с помощью гибкого диска. Однако такая скорость передачи данных по мере роста объема памяти и быстродействия ЭВМ быстро оказалась слишком малой для практического применения, и десятки фирм включились в гонку создания специального аппаратного (сетевых плат) и программного обеспечения, причем скорость передачи данных достигла десятков/сотен мегабит в секунду. Каждая фирма предлагала свой стандарт передачи данных и собственное программное обеспечение, через несколько лет были выработаны общеупотребительные стандарты на *сетевые протоколы* (наборы правил и соглашений, в соответствие с которыми производится обмен данными по сети) передачи данных.

В настоящее время поддержка наиболее распространенных сетевых протоколов встраивается непосредственно в ОС (поддержка дополнительных протоколов устанавливается по желанию пользователя), пользователю предоставляются штатные средства поддержки сетевых функций (подключение к удаленной ЭВМ, обмен файлами по сети и др.) [2]. Вместе с тем возможности современных ОС позволяют программисту разрабатывать свои собственные приложения, пользуясь средствами встроенной в ОС сетевой поддержки.

Следующий (пока полностью не реализованный) этап развития программного обеспечения сетей - создание распределенной системы (среды распределенных приложений) и соответственно распределенной ОС. Распределенную систему можно определить как систему, в которой существование нескольких автономных компьютеров является прозрачным (т.е. неощутимым) для пользователей [5]. Другими словами, одна ОС управляет несколькими сетевыми компьютерами и распределяет их ресурсы (процессорное время, оперативную и дисковую память и др.) между выполняемыми приложениями. Например, Windows'NT не является распределенной ОС (она может работать на многопроцессорном компьютере, планируя загрузку всех его процессоров, но требует от последних использования общей памяти).

Узким местом при практической реализации распределенной ОС является как теоретические (планирование рациональной загрузки процессоров различного типа при их числе, измеряемом сотнями и тысячами - трудноосуществимая задача), так и практические сложности реализации (скорость передачи данных по сети на много порядков ниже скорости обмена информацией процессоров с оперативной памятью в пределах одной ЭВМ).

Интересной разработкой в области создания сред распределенных вычислений является, например, представленный компаниями Compaq и Santa Cruz Operation программный продукт (фактически расширение ОС) Compaq ProLiant Clusters for SCO UnixWare ver.7.1, позволяющий объединять известные серверы ProLiant в кластер (сообщество компьютеров) и управлять этим кластером как единым ‘виртуальным компьютером’; при этом в кластере (в настоящее время в кластер может входить от 2 до 6 серверов) отсутствует как таковой ‘главный’ узел, а процессы могут (активно) мигрировать с одного компьютера на другой (нагрузка распределяется между узлами динамически).

При объединении компьютеров в сети вопросы несанкционированного доступа к информации (в том числе проникновение в домашние и офисные ЭВМ и проблема компьютерных вирусов) вышли на уровень ‘мирового зла’.

1.**ЦЕЛЬ РАБОТЫ**

Целью работы является дать начальные теоретические и практические понятия и сведения о программном обеспечении сетей ЭВМ (причем как на уровне штатного ПО данной ОС, так и на уровне практического создания сетевых приложений для локальных, корпоративных и глобальных сетей). При ознакомлении с пособием весьма желательна работа с указанными литературными источниками, а также практика на ЭВМ; данная основа и постоянная практика позволят программисту стать профессионалом.

В качестве основной операционной системы выбрана Windows’NT версий от 4.0 до 5.1 (W’XP). Основой этого выбора послужили серьезные сетевые возможности указанной ОС (вобравшие в себя большинство из известных на данный момент), повышенная работоспособность (надежность, способность самовосстанавливаться при сбое оборудования и др.) и совместимость Windows’NT со стандартом POSIX - *переносимый интерфейс основанных на UNIX операционных систем (Portable Operating System Interface based on uniX)*.

В целом POSIX (стандарт IEEE 1003.1-1988) поощряет фирмы, реализующие UNIX-подобные интерфейсы, делать их совместимыми, чтобы программисты могли максимально легко переносить свои приложения с одной системы на другую. Таким образом заканчивается очередной виток эпопеи

WINDOWS - от почти полного отрицания UNIX до принятия основанного на UNIX'е стандарта POSIX.

2.**СЕТИ КОМПЬЮТЕРОВ**

2.1.**ОСНОВНЫЕ ПОНЯТИЯ**

Сеть ЭВМ - комплекс аппаратного и программного обеспечения, поддерживающий функции обмена информацией между отдельно расположенными (на расстояниях от нескольких метров до тысяч километров) компьютерами. *Сеть с централизованным управлением* содержат одну или более выделенных ЭВМ (серве**о**в), управляющих обменом по сети (остальные ЭВМ в этом случае называются *рабочими станциями*), *одноранговая сеть* не содержит выделенных машин (функции управления сетью осуществляются рабочими станциями поочередно).

Соответственно *программное обеспечение компьютерных сетей* - комплекс программ, поддерживающий функции обмена информацией между отдельно расположенными ЭВМ. В настоящее время *программное обеспечение компьютерных сетей* обычно является (иногда опционально устанавливаемой) составной часть операционных систем.

Локальная вычислительная сеть (ЛВС) - система связи отдельно расположенных ЭВМ на относительно небольшом расстоянии (обычно в пределах помещении и/или этажа здания); обычно объединяет до нескольких десятков (чаще однотипных) компьютеров, физическая линия связи - двухпроводной кабель или коаксиальный кабель [3].

Корпоративная вычислительная сеть - сеть, работающая по протоколу TCP/IP и не обязательно подключенная к InterNet, но использующая коммуникационные стандарты InterNet'а и сервисные приложения, обеспечивающие доставку данных пользователям сети; эксплуатируется в пределах (крупной) организации.

Глобальная вычислительная сеть объединяет множество локальных сетей и сотни тысяч - миллионы разнотипных ЭВМ по всему миру, физическая линия связи - оптокабель или космическая радиолиния связи.

Рабочая группа (workgroup) - набор компьютеров, объединенных для удобства при просмотре сетевых ресурсов одним именем.

*Дом**е**н (domain)* - определенная администратором сети совокупность компьютеров, использующих в операционной системе WINDOWS NT Server общую базу данных и систему защиты; каждый дом**е**н имеет уникальное имя.

Узел (host) - подключенное к сети устройство (обычно компьютер), идентифицируемое собственным *адресом* (например, в сети InterNet host-адресом является уникальное 32-разрядное двоичное число, подробнее см. подраздел 7.1.1).

Скорость передачи данных по компьютерной сети измеряется в *битах в секунду* (bps - *bit per second)* или бодах (*boud*).

Трафик (traffic) - поток сообщений в *разделяемой среде передачи данных*, часто используется для грубой оценки *уровня использования передающей среды* (тяжелый, средний, легкий трафик).

Серверная ЭВМ - компьютер (обычно обладающий высоким быстродействием и значительным объемом оперативной и дисковой памяти) и выполняющий запросы, поступающие с *клиентских ЭВМ*.

Файл-сервер - выделенная ЭВМ, выполняющая функции хранения данных и программ, используемых пользователями на клиентских ЭВМ.

Серверное приложение - выполняющееся ЭВМ приложение, могущее выполнять запросы, генерируемые другим (выполняющемся на данной или удаленной ЭВМ) *приложением-клиентом*.

Клиентская ЭВМ - пользовательский компьютер (обычно обладающий ограниченными ресурсами), выдающий запросы для исполнения серверу.

Клиентское приложение - приложение, обращающееся (с целью выполнения отдельных функций) к другому *приложению-серверу* (и обычно инициирующее начало его выполнения и завершение).

Протокол (коммуникационный) - набор правил и соглашений, согласно которому взаимодействуют два (или более) компьютеров.

Топология (topology) сети - физическая конфигурация машин в сети.

Временное уплотнение при передаче данных - метод передачи данных по линии связи, основанный на последовательной (по времени) передаче пакетов (порций) данных, причем каждый пакет снабжен маркером (в состав которого входит адрес, идентифицирующий машину-получателя пакета и некоторая дополнительная информация). Временное уплотнение является стандартом для *систем коллективного пользования*, при этом множество пользователей получают высокоскоростной канал, доступный в течение всего времени (но по отношению к каждому из них канал имеет очень низкий показатель использования).

Маршрутизация - процесс определения (оптимального) пути доступа к объектам (компьютерам) сети.

Пакет (датаграмма) - определенное количество байт, сгруппированное вместе и посылаемое одновременно (практически все сети коммуникаций передают данные небольшими частями - пакетами или датаграммами).

2.2.**ИСТОРИЯ ВОПРОСА**

Пожалуй, лишь первые несколько месяцев после начала производства ЭВМ требование объединения компьютеров между собой не стояло остро. Однако даже первые ПЭВМ были оснащены аппаратурой, пригодной для создания (примитивной) сети (порты для последовательного обмена данны-

ми со скорость до 9600 бод). С развитием технических средств (сетевые платы, линии связи со значительной пропускной способностью) и соответствующего сетевого ПО процесс объединения ЭВМ в сети принял лавинный характер [1].

Существует минимум две основных причины стремительного развития компьютерных сетей.

Первая - огромные возможности ЭВМ в обмене информацией (причем информацией любого типа - от простейших посланий в виде текстовых файлов до сложных форматов медиаинформации); в большинстве случаев сетевой (в т.ч. глобальный сетевой) обмен существенно дешевле традиционных почтовых посланий и телефонных разговоров.

Вторая (не столь р**а**звитая в нашей стране) - возможность распределенных вычислений (например, использование значительных вычислительных ресурсов мощных удаленных компьютеров - к примеру, снабженных ориентированными на векторные операции процессорами).

Обе причины вызывают еще б**о**льший интерес пользователей вследствие ‘врожденных’ способностей ЭВМ по переработке и визуализации информации различного типа. Дальнейшее развитие сетевых технологий инициировано в последние десятилетия (относительно удачным) решением глобальной сети InterNet.

Однако до сих пор серьезной проблемой развития компьютерных сетей является далеко недостаточная совместимость различных операционных систем и программного обеспечения, используемых в ЭВМ различных мировых фирм-производителей (проблема остаются даже после выработки стандартов на сетевые платы и протоколы).

Дополнительная информация об истории компьютерных сетей приведена в разделе 3 данной работы.

2.3.**ОПОРНАЯ МОДЕЛЬ OSI**

В общем случае задача сетевого программного обеспечения состоит в приеме запроса (обычно это запрос ввода-вывода) от приложения на одной машине, передаче его на другую машину, выполнения запроса на удаленной машине и возврате результата на первую машину. В ходе этих операций запрос несколько раз преобразуется. Высокоуровневый запрос (например, ‘прочитать N байтов из файла X на машине Y’) требует, чтобы программное обеспечение определило, как достичь машины Y и какой коммуникационный протокол она ‘понимает’. Затем запрос должен быть преобразован для передачи по сети - например, разбит на короткие пакеты информации. Когда запрос достигнет другой стороны, необходимо проверить его целостность, декодировать и послать на выполнение соответствующему компоненту ОС. По

окончании выполнения запрос должен быть декодирован для обратной передачи по сети.

Для помощи производителям в стандартизации и интегрировании производимого сетевого ПО, Международная организация по стандартизации (ISO, *International Standart Organization*) в 1984 году определила *программную модель* пересылки сообщений между компьютерами. Эта модель получила название *опорной модели соединения открытых систем - Open Systems Interconnection* (OSI*) reference model* [1,2,5] . В модели OSI определены семь уровней программного обеспечения, как показано на рис.2.1.

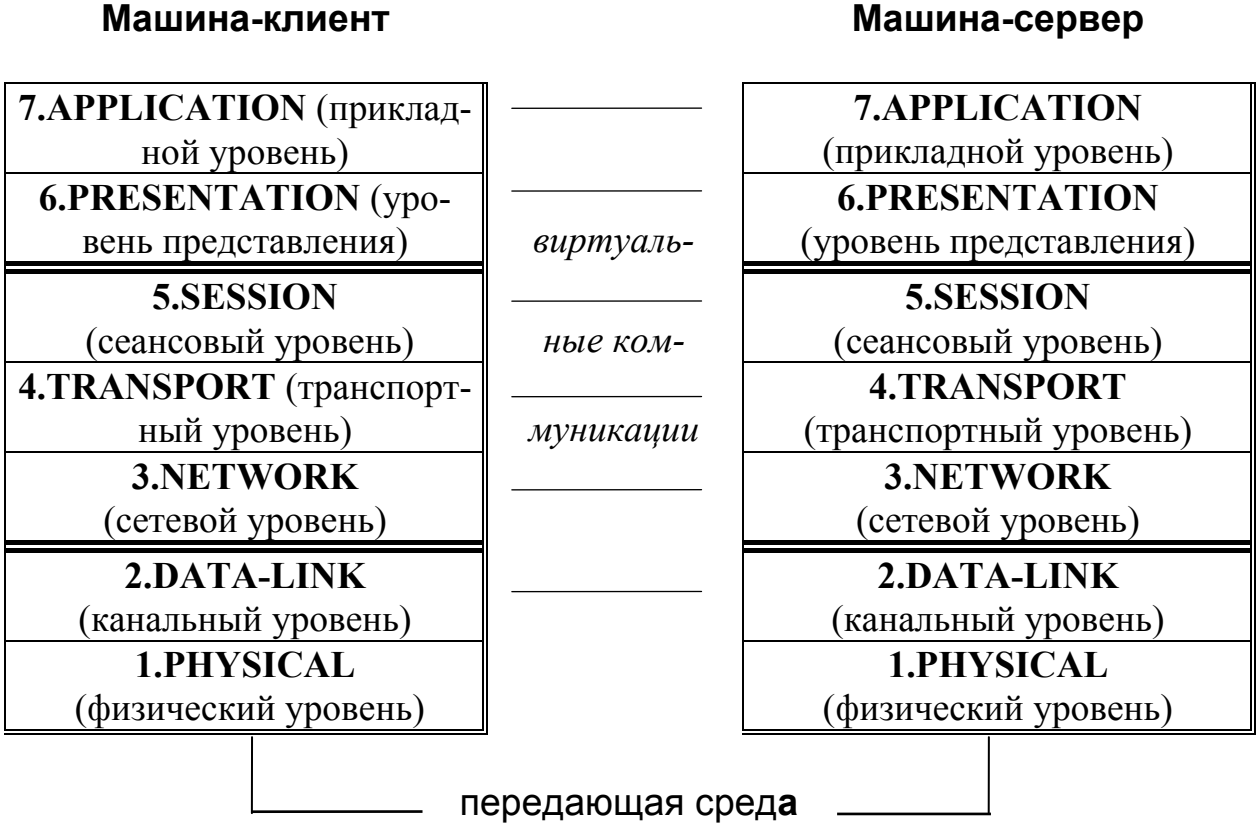

Рис.2.1. Опорная модель OSI.

Опорная модель OSI - идеальная схема, точно реализованная на очень немногих системах, однако она часто используется при обсуждении основных принципов работы сетей. Каждый уровень одной из машин ‘считает’, что он ‘разговаривает’ на одном и том же языке (или *протоколе*) с соответствующем уровнем другой ЭВМ (т.н. виртуальные связи между уровнями, условно показаны пунктиром на рис.2.1). Однако в действительности сетевой запрос должен ‘спуститься’ до самого нижнего (физического) уровня (на котором обе ЭВМ в реальности обмениваются данными), затем он передается по физическому носителю и вновь ‘поднимается’ до уровня, который его ‘поймет’ и обработает. Набор протоколов, в соответствие с которым запрос

проходит вниз по уровням сети и обратно, называется *стеком протоколов (protocol stack)*. Каждый уровень несет ответственность за выполнение ограниченного набора функций и может взаимодействовать только с двумя непосредственно прилежащими уровнями.

Задача каждого уровня состоит в предоставлении обслуживания верхним уровням, *абстрагируясь от того, каким образом* реализовано это обслуживание. Ниже приведено (краткое) описание каждого уровня модели OSI.

- Прикладной уровень. Обрабатывает передачу данных между двумя сетевыми приложениями (включая проверку прав д**о**ступа, идентификацию взаимодействующих машин и инициирование передачи данных). Большинство сетевых программ-утилит фактически являются частью именно этого уровня.
- Уровень представления. Отвечает за формирование данных (в том числе решает, должны ли строки заканчиваться парой символов ‘возврат каретки/перевод строки’ - CR/LF) или только символом ‘возврат каретки’ - CR; должны ли данные быть сжаты или закодированы и др.
- Сеансовый уровень. Управляет соединением между взаимодействующими приложениями (включая синхронизацию высокого уровня и контроль за тем, какое из приложений ‘говорит’, а какое ‘слушает’).
- Транспортный уровень. Осуществляет разбивку сообщения на пакеты и присваивает номера пакетам, чтобы гарантировать их прием в надлежащем порядке. Кроме того, изолирует сеансовый уровень влияния аппаратных изменений.
- Сетевой уровень. Отвечает за маршрутизацию, управление интенсивностью трафика и межсетевой обмен. Сеансовый уровень - наиболее высокий из уровней, ‘понимающих’ топологию сети (т.е. физическую конфигурацию машин в последней), тип физических соединений между ними и ограничения пропускной способности, длины используемых кабелей и др.
- Канальный уровень. Пересылает низкоуровневые кадры данных, ожидает подтверждения их получения и повторяет передачу кадров, потерянных в ненадежных линиях связи.
- Физический уровень. Передает (и принимает) биты по сетевому кабелю (или другой физической передающей среде).

Уровни 1 и 2 (физический и канальный) являются *уровнями аппаратных средств*; уровни 3, 4, 5 образуют *подсетевой уровень* сети, который содержит программные средства, управляющие аппаратными средствами сети. Подсетевой уровень определяет один из двух важных интерфейсов ‘прикладная программа - сеть’. Некоторые прикладные программы (особенно исполь-

зующие интенсивный обмен данными - например, коммуникационные шлюзы) присоединяются к сети на уровне 5 (сеансовом), большинство же прикладных программ присоединены к сети на уровне 6 (уровне представления). Наконец, ПО управления сетью образует уровень 7 (прикладной).

Как было сказано, уровни OSI часто неточно соответствуют реальным программным модулям (например, транспортное программное обеспечение часто ‘пересекает’ границы нескольких уровней). Фактически термин ‘транспорт’ часто используется в качестве общего обозначения всех четырех нижних уровней, а расположенные на трех верхних уровнях компоненты именуют ‘пользователями транспорта’.

В качестве примера на рис.2.2 представлен общий вид сетевых компонентов Windows’NT, их соответствие уровням модели OSI и используемые различными уровнями протоколы [5].

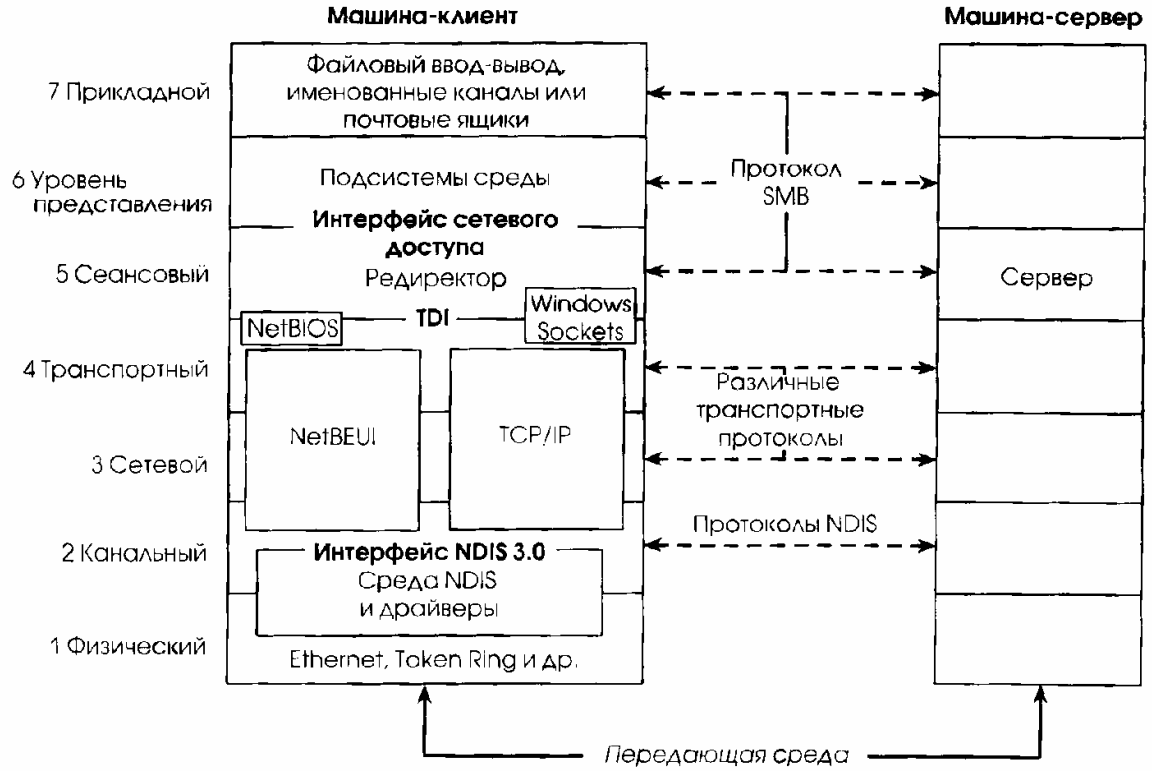

Рис.2.2. Сетевые компоненты Windows’NT, их соответствие уровням модели OSI и используемые различными уровнями протоколы.

2.4. ОСНОВНЫЕ ПРОТОКОЛЫ, ПРИМЕНЯЕМЫЕ В КОМПЬЮТЕРНЫХ СЕТЯХ

Как известно, *протокол* суть совокупность правил, регламентирующих процедуру коммуникации.

Уточним, что различают *протоколы физического уровня* (Ethernet, Token-Ring, ARCnet и т.д.), определяющих именно физические правила сетевых соединений (уровень, полярность, длительность и т.п. сигналов) и реализуемые сетевыми картами и иной сетевой аппаратурой, и *протоколы высоких* (примыкающих к транспортному и выше) *уровней* (XNS, IP/TCP и т.д.), определяющие логическую структуру сообщений и реализуемые в основном программным путем. Так как транспортные протоколы определяются и поддерживаются именно сетевым ПО, представляет интерес дать некоторые сведения по этому вопросу.

Например, фирма Microsoft Corp. штатно предоставляет следующие сетевые транспортные протоколы [5,9] :

- транспорт NetBEUI (*NetBios Extended User Interface transport*) - транспортный протокол локальной сети, созданный для работы совместно с сетевым интерфейсом NetBIOS фирмы Microsoft Corp.
- транспорт TCP/IP (*Transmission Control Protocol / Internet Protocol transport*) - разработанный для Министерства обороны США протокол, предназначенный для соединения разнородных систем через глобальные сети. TCP/IP широко распространен в сетях UNIX и позволяет WINDOWS'NT взаимодействовать с различными сервисами на UNIX-машинах.

Заметим [13], что протокол TCP/IP фактически представляет собой два различных протокола, работающих совместно - не гарантирующий доставку пакетов данных по сети протокол IP *(Internet Protocol*) и гарантирующий доставку пакетов в правильной последовательности протокол TCP (*Transmission Control Protocol*), в свою очередь протокол TCP/IP может служить носителем ('оберткой') для других протоколов - например, для протоколов IPX, NetBIOS, служебных протоколов адресации ARP (*Address Resolution Protocol*) и протокола межсетевых управляющих сообщений ICMP (*Internet Control Message Protocol*). В локальной сети TCP/IP-пакеты упаковываются в 'обертку' пакетов Ethernet, сами же TCP/IP-пакеты являются 'оберткой' для HTTP.

Протокол SLIP (*Serial Line Internet Protocol*) позволяет изолированным компьютерам связываться с TCP/IP через телефонную сеть. Этот протокол определяет метод разбиения датаграмм на фреймы при передаче их по последовательному каналу и указывает конец одной и начало другой датаграммы. Хотя протокол SLIP вполне подходит для установления связи с дисковым набором, но недостатки в адресации, идентификации типа и сжатии данных делают его негибким, медленным и трудным в конфигурации [6].

Межузловой протокол PPP (*Point-to-Point Protocol*) был разработан для устранения недостатков SLIP; для PPP разработано несколько расширений,

таких как опция предоставления имен серверов (DNS, *Domain Names Service - служба доменных имен*, подробнее см. раздел 7), обеспечение безопасной идентификации пользователя и объединение многочисленных соединений в одно логическое соединение с повышенной полосой пропускания [6].

Среди прочих существующих или находящихся в стадии разработки протоколов разработки фирмы Microsoft Corp. и другими фирмами можно назвать

- IPX/SPX (*Internet Packet eXchange / Sequest Packet eXchange*) - набор транспортных протоколов, используемых программным обеспечением NetWare фирмы Novell Corp. [3,10].
- DECnet - используемый фирмой Digital Equipment Corp. транспортный протокол, предназначенный для связи систем Windows'NT с сетями DECnet.
- AppleTalk - разработанный фирмой Apple Corp., Inc протокол для взаимодействия WINDOWS'NT с компьютерами Apple Macintosh.
- XNS (Xerox Network Systems) - транспортный протокол, разработанный фирмой Xerox Corp. и использовавшийся в первых сетях Ethernet.

В InterNet часто применяются следующие протоколы [6] :

- TIME - наиболее простой протокол, с помощью которого извлекаются данные времени из соответствующего сервера.
- Протокол FTP (*File Transfer Protocol*), мало изменившийся с начала 70-х годов, работает на всех компьютерах (от платформ PC DOS до суперкомпьютеров); данный протокол считается устаревшим, т.к. не может взять на себя приоритет управления трафиком и возобновлять несостоявшиеся (аварийно завершенные) передачи данных.
- Протокол SMTP (*Simple Mail Transfer Protocol*) известен с 1980 года и был рассчитан на обмен почтой между 'большими' ЭВМ (*mainframe - мэйнфрейм*), которые имеют постоянное соединение (но не на имеющие случайное, непостоянное соединение ПЭВМ) .
- NNTP (*Network News Transfer Protocol*) - относительно сложный протокол, служит для передачи новостей между серверами новостей и от сервера к клиенту.
- Протокол POP (*Post Office Protocol*) снял 1984 году ограничения протокола SMTP путем добавления двух новых функций - восстановления всех сообщений (в случае аварии) и удаления их с сервера (при успехе передачи). Текущая версия, POP3, добавляет несколько новых характеристик, сохраняя многое из структуры первоначальной версии. Однако и SMTP и POP поддерживают только поток текста ASCII и не стандартизируют обмен данными современных форматов.

- Формат MIME (*Multipurpose Internet Mail Extension*) появился в 1992 году и снял ограничения SMTP и POP в области передачи двоичных файлов (графика, мультимедиа и др.); промежуточные (взятые из ОС UNIX) преобразования ‘формат ASCII ⇔ двоичный формат’ UUEncode и UUDecode в было время широко применялись в FidoNet.
- HTTP (*HiperText Transfer Protocol*) - протокол передачи гипертекста в InterNet [6,13]; информацию о самой последней версии HTTP можно получить непосредственно от рабочей группы HTTP по адресу www.ics.uci.edu/pub/ietf/http.

Желающие могут ознакомиться со всеми тонкостями существующих протоколов в виде документов RFC (*Request For Comment*) на сервере www.cis.ohio-state.edu/htbin/rfc (например, протоколы TIME, SLIP и PPP описаны в документах RFC868, RFC1055 и RFC1171 соответственно, документы RFC1251 и RFC1252 описывают формат MIME, документ RFC1867 определяет прием файлов в виде HTML-документов и т.д.), список используемых в InterNet протоколов содержится в файле PROTOCOL операционной системы WINDOWS.

Не следует считать, что существующие протоколы исчерпывают все возможности сетевого обмена данными. Любой (подготовленный) разработчик ПО может предложить собственный (позволяющий успешно выполнять некоторые специфические действия) протокол; в случае действительного востребования этой разработки она будет признана стандартом (и включена в список документов RFC).

Например, в настоящее время используется версия 1.1 протокола HTTP. Ее поддерживают все основные броузеры и WEB-серверы. Протокол HTTP 1.1 описан в RFC-2068 и превосходит предыдущую версию HTTP 1.0 – прежде всего, по производительности. Однако, есть и другие отличия, описанные ниже

- Постоянные соединения. Протокол HTTP 1.1 устанавливает меньше TCP-соединений, чем HTTP 1.0. Версия 1.0 устанавливает и разрывает TCP-соединение для каждого HTML-запроса, а HTTP 1.1 создает TCP-соединение, сохраняющееся на протяжении многих запросов.
- Протокол HTTP 1.1 поддерживает сжатие данных. Это означает, что файлы между клиентом и сервером могут передаваться сжатыми, что снижает нагрузку на сеть.
- Протокол HTTP 1.1 поддерживает многие языки сетевого программирования.
- Создание виртуальных хостов. Протокол HTTP 1.1 позволяет одному WEB-серверу иметь несколько доменных имен. В настоящее время эта

ситуация распространена широко (например, когда поставщик услуг InterNet'а поддерживает несколько доменов).

Консорциум W3C работает над протоколом HTTP-NG (*Next Generation*), который, как предполагается, заменит HTTP. К HTTP-NG предъявляются следующие требования:

- Простота – протокол HTTP-NG должен быть прост для реализации и обслуживания.
- Расширяемость – на случай ситуации, не предусмотренной в процессе разработки.
- Масштабируемость – вне зависимости от того, используется ли HTTP-NG в маленькой локальной сети или в сети InterNet.
- Эффективность – ожидается, что протокол HTTP-NG будет намного эффективнее HTTP. Последний плохо работает в сетях с большим временем задержки. Причина в том, что HTTP – протокол одиночных запросов и ответов. Кроме того, он перегружен информацией. Протокол HTTP-NG призван устранить эти и другие недостатки.

В последнее десятилетие наметилась тенденция разработки формальных методов описания протоколов, значительно упрощающих разработку и тестирования новых протоколов передачи данных (см. подраздел 3.2), требования обеспечения конфиденциальности передаваемой по сетям информации инициировали разработку новых протоколов обеспечения секретности данных (см. подраздел 8.2).

2.4.1. КРАТКОЕ ОПИСАНИЕ КОМАНД РАСПРОСТРАНЕННЫХ ПРОТОКОЛОВ

В качестве иллюстрации уточним принципы функционирования протоколов FTP, SMTP, POP, NNTP и HTTP, более подробные описания (в т.ч. других протоколов) приведены в работе [6].

Команды FTP основаны на тексте, и пользователь может ввести их с помощью командной строки клиентского приложения FTP.EXE (находится в подкаталоге /SYSTEM при инсталлированной ОС WINDOWS).

Все команды протоколов FTP завершаются стандартным сочетанием символов возврата каретки и новой строки (строка '\r\n' в стиле C), ответ каждой из них содержит число из трех цифр в качестве первого фрагмента информации в ответе. Первая цифра кода ответа определяет характер ответа (положительное или отрицательное завершение выполнения команды и др.), вторая цифра в кодах ответа показывает функциональную область, в которой

выполнялась команда, третья цифра уточняет содержащуюся в ответе информацию.

Ниже приведены некоторые (характерные) команды протокола FTP (всего их несколько десятков)

- ***CWD*** *- изменить рабочий каталог*. Синтаксис команды следующий:
  CWD путь \r\n
  Возможные коды откликов: 250, 421, 500, 501, 530, 550 (ниже приведены отдельные описания кодов отклика).
- ***CDUP*** *- переход в родительский каталог.*
  CDUP \r\n
  Возможные коды откликов: 200, 421, 500, 501, 530, 550.
- ***PASS -*** *пароль.*
  PASS пароль \r\n
  Возможные коды откликов: 202, 230, 32, 421, 500, 501, 520.
- ***TYPE*** *- указывает тип данных для передачи - A (ASCII) и I (Image).*
  TYPE тип_кода \r\n
  Возможные коды откликов: 200, 421, 500, 501, 504, 530.
- ***LIST*** *- прочитать список содержимого текущего (или указанного) каталога (тип данных - ASCII).*
  LIST [путь] \r\n
  Возможные коды откликов: 125, 150, 226, 250, 421, 425, 426, 450, 451, 500, 501, 502, 530.
- ***MKD*** *- создать каталог в файловой системе сервера* (доступна только привилегированному пользователю).
  MKD [путь] \r\n
  Возможные коды откликов: 257, 421, 500, 501, 502, 530, 550.
- ***RETR*** *- отправить конкретный файл клиенту.*
  RETR имя_файла \r\n
  Возможные коды откликов: 110, 125, 150, 226, 250, 421, 425, 426, 450, 451, 500, 501, 502, 530, 550.
- ***HELP*** *- возврат информации подсказки обо всех командах или о конкретной команде.*
  HELP [командная_строка] \r\n
  Возможные коды откликов: 211, 214, 421, 500, 501, 502.
- ***QUIT*** *- выход из системы.*
  QUIT \r\n
  Возможные коды откликов: 222, 500.

Полная таблица описания кодов отклика на команды FTP содержит около 40 описаний, ниже приведены лишь некоторые из них

Код	*Значение*

200	OK (обобщенный положительный ответ на команду)
202	Данный узел не реализует команду
421	Служба отсутствует, поэтому соединение закрывается
500	Синтаксическая ошибка, команда не опознана
501	Синтаксическая ошибка в аргументах или параметрах команды
502	Команда не реализована
530	Пользователь не зарегистрирован
550	Запрашиваемое действие не было выполнено (файл не обнаружен или в доступе было отказано)

Так же как протокол FTP, протоколы SMTP, POP, NNTP и HTTP (последний, например, включает всего 3 высокоуровневых метода - HEAD, GET, POST) осуществляют обмен клиентской машины с серверной подобным образом, поэтому достаточно квалифицированному программисту не представляет трудностей самостоятельно организовать требуемый диалог с применением указанных протоколов.

Наиболее подробную (и самую современную) информацию можно получить из документов RFC (а для протокола HTTP по InterNet-адресам www.w3c.org, info.cern.ch или www.ics.uci.edu).

2.5. КЛИЕНТ-СЕРВЕРНАЯ МОДЕЛЬ И РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛЕНИЯ

Компоненты сетевого ПО являют собой типичный пример клиент-серверных приложений - в каждый момент времени один из процессов (выступающий при этом в роли *клиента*) запрашивает некоторые сервисные функции (например, требование связаться по сети с удаленной ЭВМ) у другого, последний же предоставляет требуемый сервис (является *сервером*); во многих случаях в зависимости от ситуации процессы могут меняться функциями.

Заметим, что процесс-клиент и процесс-сервер могут исполняться как на одной и той же ЭВМ, так и на разных (соединенных) сетью ЭВМ. Каждая многозадачная ОС имеет специфические средства обеспечения взаимодействия ‘клиент-сервер’ [2,5,7].

Итак, основная идея модели ‘клиент-сервер’ состоит в разделении ПО на несколько процессов, каждый из которых реализует специфический набор сервисов: например, распределение памяти, создание процесса или планирование процессов. Каждый сервер (*server*) выполняется в *пользовательском режиме*, проверяя в цикле, не обратился ли к нему с требованием обслуживания какой-либо клиент (*client*). Клиент (которым может быть другой компонент ОС) или прикладная программа, запрашивает выполнение *сервиса*, посылая серверу сообщение. Ядро ОС (выполняющееся в *режиме ядра*), доставляет сообщение серверу; последний выполняет запрашиваемые действия,

после чего ядро ОС возвращает клиенту результаты в составе другого сообщения.

Преимущество построения ПО на принципе клиент-серверного подхода состоит в автономизации отдельных компонентов ПО - каждый компонент имеет ограниченный размер, выполняется как отдельный процесс пользовательского режима, авария (приводящая, возможно, к перезапуску процесса) одного из них не нарушает работы остальных компонентов ПО. Именно на таком принципе построена известная операционная система WINDOWS фирмы Microsoft Corp.

Важно, что различные серверные процессы могут выполняться на различных процессорах многопроцессорного компьютера или даже на разных компьютерах, что делает построенную на клиент-серверной основе ОС пригодной для распределенных вычислительных сред. В случае распределенной ОС клиентам даже не требуется знать, обслуживается их запрос локально (на данной ЭВМ) или удаленно.

В Windows'NT средства обмена сообщениями между распределенными (часто выполняющимися на различных компьютерах в сети) приложениями обеспечивает (путем поддержки сетевого транспорта и защиты) *служба сетевого обмена данными* (DDE*, Dynamic Data Exchange*); также поддерживается *модель выполнения распределенных приложений* в сети (DCOM, *Distributed Component Object Model*). Ощутимый недостаток DCOM состоит в том, что каждый клиент может взаимодействовать только с одним конкретным компьютером, имеющим в составе ПО нужный сервер приложений (сколько бы их ни было в сети); при этом информационная система не имеет никакой защиты от сбоев, вызванных перегрузкой или отказом сервера приложений (так как отсутствует возможность переключения клиентского приложения между несколькими серв**е**рами сети); этот недостаток устранен в технологии OLEnterprise фирмы Inprise Corp. [7].

В состав Windows'NT изначально введены средство *локального вызова процедур (*LPC, *Local Procedure Call)* - оптимизированного механизма исполняющей системы NT для локальной передачи сообщений серверной процедуре и средство *удаленного вызова процедур (*RPC, *Remote Procedure Call)* - механизм вызова процедур с удаленной машины (именно через RPC взаимодействуют клиенты и сервер**а** приложений согласно технологии OLEnterprise). Не менее интересным являются введенные в Windows'NT средства передачи данных между процессами (в том числе выполняющимися на различных ЭВМ, объединенных сетью) - именованные и анонимные каналы передачи данных, каналы типа Mailslot [14].

Таким образом, хотя Windows'NT и не является в полном смысле слова распределенной ОС, фирма-разработчик Microsoft Corp. серьезно подготовилась к реальному созданию распределенной ОС, и она будет (с б**о**льшим или м**е**ньшим успехом) разработана в свое время.

2.6. ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ И КЛАСТЕРЫ КОМПЬЮТЕРОВ

В последние десятилетия существенно возрос интерес к решению ‘больших задач’ (моделирование климата, задачи обтекания летательных аппаратов, механики твердого тела, генетического моделирования и др., [24]), требующих огромной (до 10^{15} операций с плавающей точкой в секунду - Flops) производительности компьютеров. Однако подобные ЭВМ чрезвычайно дороги и быстро устаревают. Один из подходов к созданию *масштабируемых* (допускающих возможность укрупнения) вычислительных систем - создание кластеров (сообщество вычислительных узлов, объединенных коммуникационной средой); на кластерах естественным образом реализуется *распараллеливание* вычислительных задач. Обзор состояния технологий параллельных вычислений достаточно полно изложен в работе [24].

В качестве вычислительных узлов обычно применяются относительно недорогие микропроцессорные системы на основе Intel Pentium IV-V, i860, DEC Alpha, для создания коммуникационной среды используют известные сетевые технологии. Одним из известных проектов такого рода явилось семейство Cray T3D/T3E (более 2000 процессоров, см. www.cray.com). Созданная в Межведомственном Суперкомпьютерном Центре (МСЦ, www.jscc.ru) система МВС 1000М (189-й номер в списке Top500 на ноябрь 2003 г., www.top500.org) имеет пиковую производительность до 10^{12} Flops (384 двухпроцессорных вычислительных модулей с пиковой производительностью 2,7 GFlops на double-данных; каждый включает 2 процессора Alpha 21264A, 667 MHz с L2 кэш-памятью 4 Mb, 2 Gb разделяемой оперативной памяти, жесткий диск 20 Gb; при этом модули объединены межпроцессорной сетью Myrinet 2000 со скоростью обмена до 2 Gbit/sec и сетью Fast Ethernet с пропускной способностью 100 Mbit/sec). В данном случае компьютерная сеть выступает в качестве связующего звена между узлами *решающего поля*; обеспечивающим технологию распараллеливания вычислений является специализированное ПО (в настоящее время распространена система программирования MPI - *Message Passing Interface*, www.mpi-forum.org).

Крайним случаем распараллеливания вычислений является *метакомпьютинг* - организация вычислений на наиболее мощных (локальных, корпоративных, глобальных) компьютерных сетях (обычно подразумевается InterNet). Самыми известными проектами метакомпьютинга являются SETI@home (*Search for Extraterrestrial Intelligence*, setiathome.ssl.berkeley.edu, для участия в проекте зарегистрировалось около 4 млн. человек), Distributed.net (www.Distributed.net), GIMPS (*Great Inter-*

net Mersenne Prime Search, mersenne.org), Globus (www.globus.org); подробнее см. [24], подраздел 7.4 данной работы и др.

3.МЕСТО СЕТЕВОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СРЕДИ СИСТЕМНОГО И ПРИКЛАДНОГО ПО

3.1.СЕТЕВОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ОБЩЕГО НАЗНАЧЕНИЯ

Задача сетевого программного обеспечения состоит в приеме запроса (обычно это запрос на ввод-вывод) от приложения на одной машине, передаче его на другую машину, выполнения запроса на удаленной машине и возврате результата на первую машину. Таким образом, сетевое ПО может быть выполнено как в виде отдельных модулей (устанавливаемых на ЭВМ при необходимости), так и в виде компонентов самой ОС (возможно, опциональных - т.е. выбираемых при инсталляции ОС). Фактически эта последовательность и была повторена в ходе развития сетевого ПО; в настоящее время фактически все ОС включают штатные компоненты сетевого ПО.

Начало истории сетей Microsoft было положено в MS-DOS версии 3.1; в ней к файловой системе FAT были добавлены необходимые расширения блокировки файлов и записей, которые обеспечили возможность работы с файлами MS-DOS сразу нескольким пользователям. Одновременно с выходом в 1984 году MS-DOS версии 3.1 Microsoft выпустила продукт под названием Microsoft Networks, получивший неформальное название MS-NET.

MS-NET установил de-facto ряд традиций, которые позже были перенесены в Microsoft LAN Manager (несмотря на амбициозное наименование Microsoft LAN Manager ‘сетевой ОС’, на самом деле это набор сложных программ и драйверов, добавляющих сетевые возможности к существующим ОС, в частности, к MS-DOS, OS/2 и UNIX, а потом и к Windows’NT).

Например, в случае выдачи пользователем или приложением запроса на ввод-вывод для удаленного файла, каталога или принтера система MS-NET определяла эту ситуацию и направляла запрос компоненту MS-NET, называвшемуся *редиректор (redirector)*. Редиректор MS-NET принимал запрос и посылал (перенаправлял - - *redirect*) его на удаленный сервер.

Другой особенностью MS-NET являлся встроенный протокол SMB (*Server Message Block*), являющийся высокоуровневой спецификацией формата посылаемых по сети сообщений. Для пос**ы**лки имеющих формат SMB запросов на другой компьютер используется API (*Application Program Interface*) под названием *интерфейс NetBIOS (NetBIOS interface)*; впоследствие протокол SMB и API NetBIOS были использованы в многочисленных сетевых продуктах, в том числе и в Windows’NT.

Последнее, что было реализовано в MS-NET - это с*етевой сервер (network server)* - находящееся на удаленном компьютере программное обеспечение, превращающее его в выделенный файл-сервер (или сервер печати). Это ПО просто контролировало сетевое соединение и ждало поступления по нему данных в формате SMB, затем оно распаковывало их, определяло и выполняло запрашиваемую операцию (например, чтение файла), после чего посылало результат выполнения операции обратно в виде другого сообщения SMB.

Система MS-NET также включала набор утилит (со своим синтаксисом командной строки) для доступа к удаленным дискам и принтерам (например, многим известны команды формата NET USE X: \\SERVER\SHARE); до сих пор начинающиеся с символов '\\' имена обозначают сетевые ресурсы и называются именами *единого соглашения об именовании (*UNC, *Uniform Naming Convention*).

Серьезным шагом к интеграции сетевого ПО в ОС явилась система NetWare фирмы Novell, называемая (с б**о**льшей или м**е**ньшей долей истинности) сетевой ОС. В настоящее время среда NetWare способна поддерживать рабочие станции, управляемые MS-DOS, Windows, OS/2, UNIX, Mac System 7 и др., обладает развитой системой защиты данных (например, уровень привилегий д**о**ступа может быть указан отдельно для каждого каталога).

NetWare включает *систему защиты при отказах оборудования* (SFT, *System Fault Toleran*t) трех уровней

1. Дублирование на том же диске критических данных (особенно каталогов и таблиц распределения ресурсов).
2. Зеркальное копирование диска (использование два идентичных жестких диска - исходный диск и его зеркальную копию) и дублирование (полное дублирование всего искового оборудования - контроллера диска, кабеля, дисковода и носителя). Кроме этого, первый и второй уровни включают систему защиты данных, называемую *трассировкой изменений* (создание дубликата индекса базы данных до завершения *транзакции*).
3. Дублирование сетевого сервера (второй сервер находится в горячем резерве и вступает в строй при аварии первого).

Большинство этих использованных в NetWare компонентов системы защиты применены (и получили дальнейшее развитие) в Windows'NT.

ОС NetWare включает несколько нужных в практической деятельности дополнительных компонентов сетевого ПО (например, ПО обслуживания мостов - устройств для связи локальных сетей с одинаковыми или различными методами д**о**ступа), что иногда приводится в качестве обоснования (претенциозного) наименования NetWare 'сетевой ОС'.

Однако в системах NetWare и Windows'NT службы управления каталогами построены в соответствие с принципиально разными принципами. В NetWare 4.1 используется *Netware Directory Service* (NDS), позволяющая представить сеть в виде древовидной структуры, служба же управления каталогами в сетях Windows'NT представляет сеть в виде набора доменов, состоящих в *доверительных отношениях*. Обе службы позволяют централизованно управлять сетью со многими серверами. При этом пользователю, однократно зарегистрировавшемуся в сети, предоставляется возможность соединения с различными серверами. В NDS, например, удобнее просматривать все ресурсы сети, логически переносить пользователя из одной части дерева в другую, доменная же система позволяет более гибко настраивать отношения между доменами (домен может иметь как полную, так и частичную информацию о другом домене, либо вообще никакой).

Поэтому для достижения универсальности и производительности часто совместно используют NetWare и Windows'NT Server; при этом NetWare используется для работы с файлами и обслуживания устройств печати, а Windows'NT - для обмена сообщениями и работы серверов приложений (например, СУБД) на различных приложениях.

В одной из самых распространенных ОС для IBM PC-совместимых ЭВМ Windows (установленной приблизительно на 80% всех 200 млн. IBM PC в мире) сетевые компоненты встроены в саму ОС. Большинство из них написаны на языке C/C++ и выполнены в виде DLL-файлов и динамически загружаемых драйверов, что делает их частью исполнительной системы ОС. В Windows'NT применена послойная модель драйверов ввода-вывода (напоминающая естественное расположение сетевых протоколов OSI один над другим); такая структура обеспечивает модульность сетевых компонентов и создает эффективные переходы от уровня редиректора или сервера вниз к транспортным и физическим уровням сети [2,5].

На рис.3.1 показана укрупненная блок-схема операционной системы Windows'NT, на рисунке видны (как общесистемные, так и сетевые) компоненты данной ОС.

На рис.3.2 показано взаимодействие редиректора и сервера через протокол прикладного уровня SMB (*Server Message Block*); для посылки имеющих SMB-структуру данных на другой компьютер используется интерфейс NetBIOS (*NetBIOS interface*).

Фирма Microsoft Corp. в начале 21 века выпустила основанную на ядре NT операционную систему Windows XP (реально NT'5.1), призванную заменить последний вариант Windows на базе ядра Windows'9x и носящий фирменную марку MILLENNIUM; XP оснащена технологией полностью автоматической настройки ЭВМ для работы в сети, идентификации устройств стандарта *Universal Plug and Play* и новым упрощенным пользовательским интерфейсом в стиле WEB (что, впрочем, уже было испробовано в

Windows'98 и далеко не всем 'пришлось по вкусу'). Широко разрекламированная ОС Whistler также должна обладать всеми этими свойствами. Заметим, что всеми перечисленными возможностями Unix-подобные ОС обладают практически 'с самого рождения'.

Таким образом, компоненты сетевого ПО, появившиеся вначале как (необязательные) дополнения и расширения существующих ОС, в современных ОС являются встроенными (пока опциональными) компонентами системного ПО.

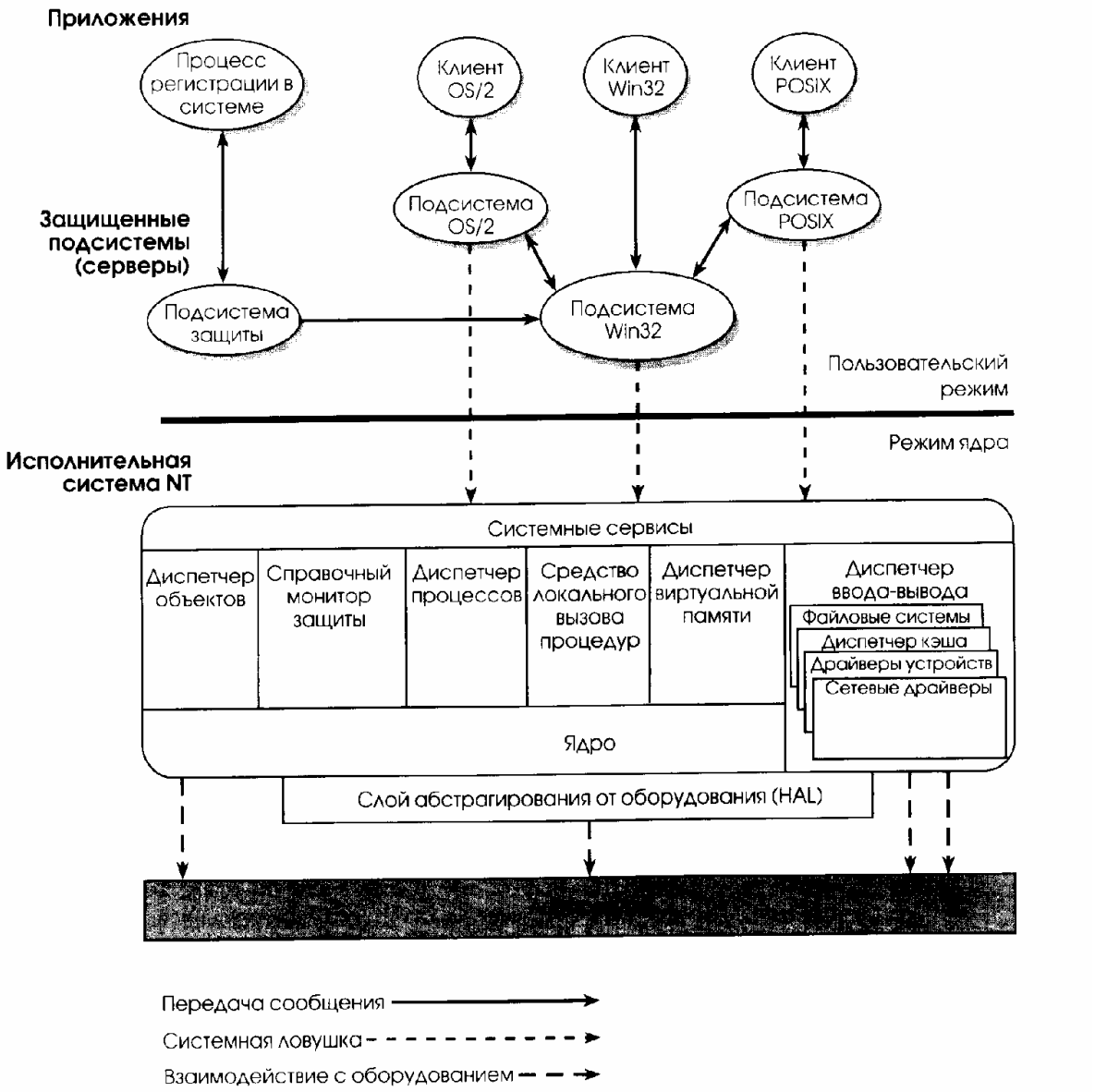

Рис.3.1. Блок-схема операционной системы Windows'NT (подсистемы эмуляции 16-разрядной Windows и MS-DOS условно не показаны).

Сетевое ПО является достаточно сложным программным обеспечением, к тому же надежность его функционирования зависит от корректности рабо-

ты физической компоненты сети (сетевые карты, линии связи), поэтому важной частью сетевого ПО являются подсистема анализа показателей и поиска неисправностей в сетях ([4] и др.).

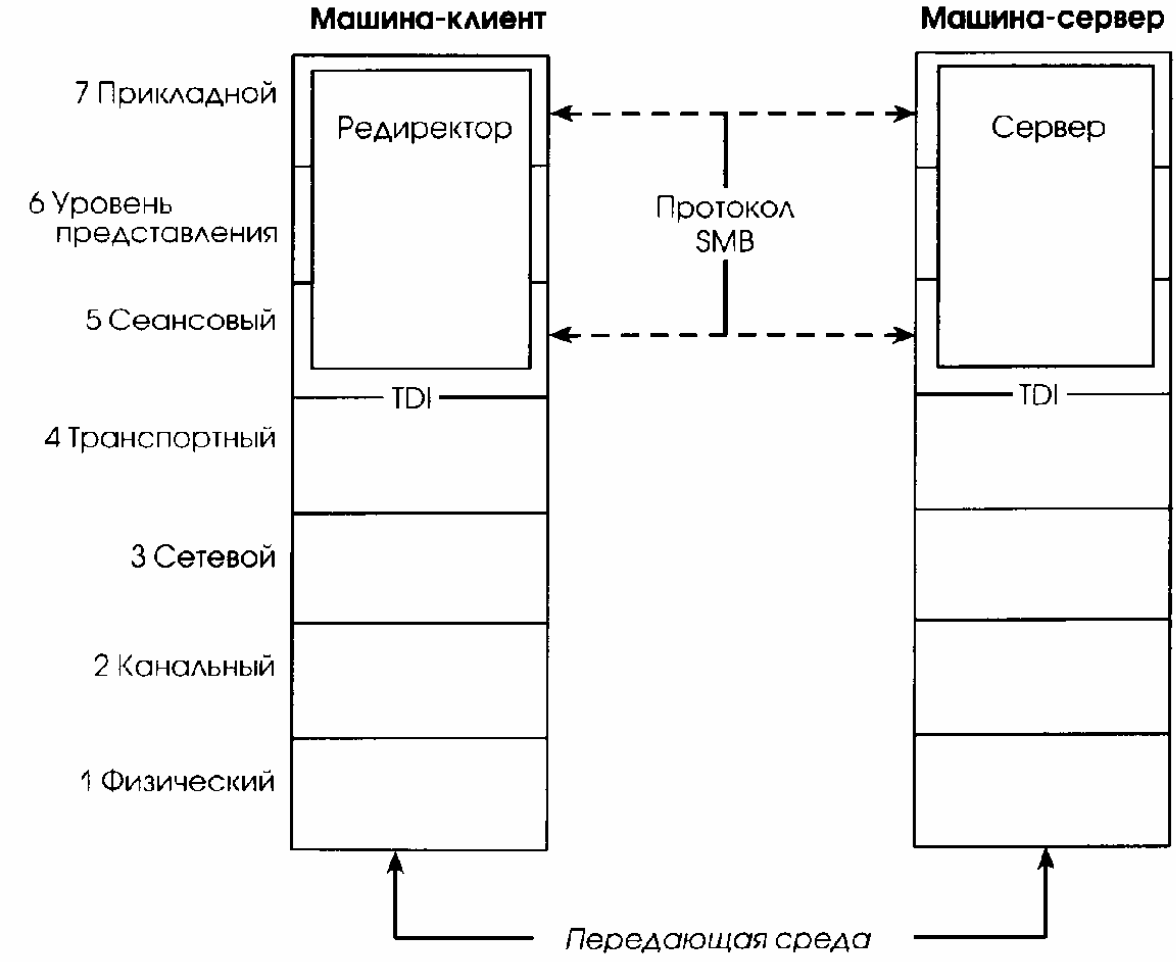

Рис.3.2. Взаимодействие редиректора и сервера при сетевом взаимодействии по протоколу SMB в среде Windows'NT.

После (корректной) установки сетевых компонентов для д**о**ступа к дисковым массивам удаленных компьютеров по сети можно использовать следующую командную строку

\\сетевое_имя_компьютера\локальный_путь_к_файлу\имя_файла

Другой вариант - использование удаленных дисковых систем с применением *маппированных(mapping) имен дисков* (отображение имен дисковых наборов в продолжение логических имен дисков данной локальной машины), при этом (с точки зрения пользователя) д**о**ступ к удаленным томам осуществляется в точности таким же образом, как и д**о**ступ к дисковым томам данной машины.

Информация о некоторых (распространенных) пакетах сетевого ПО уровня пользователя приведена ниже в разделе 7.

Следует отметить наметившуюся в последние годы тенденцию переноса части сетевого ПО на аппаратный уровень (например, концепция ‘сетевого компьютера’ фирмы Oracle, поддержанная другими фирмами мирового уровня).

3.2. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПОИСКА НЕИСПРАВНОСТЕЙ В СЕТЯХ, АНАЛИЗА И МОДЕЛИРОВАНИЯ СЕТЕЙ

Устойчивая работа современных организаций существенно зависит от устойчивости функционирования вычислительной сети, при этом роль диагностического сетевого ПО и оборудования возрастает. На первом этапе анализа обычно используются штатно входящие в состав ОС утилиты *ping, tracert, netstat, route, arp, hostname, ipconfig, nbtstat, netstat* и др.

Важную информацию о производительности сети дают *анализаторы протоколов* (аппаратно-программные устройства, физически подключаемые к сети с целью перехвата и анализа циркулирующей в сети информации, причем часто функцию такого устройства выполняет сам сетевой компьютер с соответствующим программным обеспечением). Анализаторы протоколов позволяют анализировать параметры работы сети на любом уровне начиная с физического (от анализа обрыва до выявления *сетевых коллизий* и анализа сети на уровне протоколов верхнего уровня). К программным средствам тестирования сетей относятся пакеты LANalyzer for NetWare, LANalizer for Windows и LANTEST (фирма Novell), утилита SysInfo (Symantec) и др. [3,4].

Анализ функционирования сети - трудоемкий процесс, требующий высокой квалификации персонала, однако затраты на анализ сети обычно (для серьезных сетевых применений) себя оправдывают (вследствие повышения устойчивости работы сети).

Интересным (и полезным) классом ПО являются пакетные анализаторы (снифферы, от *sniffer - вынюхиватель*) - утилиты, осуществляющие просмотр и анализ содержимого сетевого трафика в заданном сегменте сети; подобное ПО может использоваться, например, для ‘отлова’ сообщений с заданными ключевыми словами (закодированные сообщения требуют последующей расшифровки). К простым ПО подобного класса относится, например комплект SpyNet (simik.lgg.ru/spynet312.exe); в штатную поставку Windows’NT 4.0 Server входит утилита Network Monitor (устанавливается добавлением сервиса Network Monitor Tools & Agent). Функции сниффера естественно поддерживаются североамериканским проектом ЭШЕЛОН (анализ содержимого линий связей Европы) и пресловутой отечественной системой

СОРМ (тотальное протоколирование трафика Сети, подробнее см. www.ice.ru/libertarium/sorm).

Примерами программ анализа и моделирования вычислительных сетей служат COMNET III (фирма CACI Products Company, www.caciasi.com) и OPNET(OPNET, www.mil3.com). Подобные системы используют для задания исходных данных проблемно-ориентированные языки (например, MODSIM или SIMSCRIPT) с графическими расширениями; имеются библиотеки моделей протоколов и аппаратных сетевых средств. При моделировании выявляются ‘узкие’ места, задержки в передаче данных, загрузка линий, длины очередей, пиковые нагрузки; в конечном итоге обеспечивается возможность сравнения различных архитектур построения сетей, определять размещение серверов, рассчитывать трафик.

К пакетам интерактивного проектирования сетей относится NetSuit Advanced Professional (фирма NetSuit Development); система позволяет изображать поэтажную схему здания с размещенными компьютерами и сетевым оборудованием, проверять допустимость их совместного использования и иные ограничения.

3.3. ФОРМАЛЬНЫЕ МЕТОДЫ ОПИСАНИЯ ПРОТОКОЛОВ

Число эксплуатируемых в настоящее время протоколов обмена данными велико (см. подраздел 2.4); при этом разрабатываются все новые протоколы, обеспечивающие лавинное развитие сетевых технологий (появилась новая область вычислительной техники, называемая ‘протокольной технологией’).

Классическое (неформально-словесное, например, ранее упомянутые RFC-документы) описание протокольных соглашений имеет ряд недостатков; важнейшие из них - не позволяющая однозначно согласовывать разрабатываемые стандарты субъективная природа восприятия словесных описаний (следствие - описания не имеют полноты и основы для анализа), возникают трудности и труднолокализируемые ошибки при создании реализующих эти протоколы программных и аппаратных средств [8].

По сравнению со словесными формальные описания обладают существенными преимуществами - они строги и однозначны, лежащие в основе конкретного метода формального описания модели позволяют выполнить анализ (верификацию) описаний, а также автоматизировать процесс трансляции этих описаний непосредственно в машинную реализацию.

Формальные методы описания протоколов могут быть разбиты на две группы - методы первой группы рассматривают объект как автомат (т.н. ‘автоматные методы’), методы второй группы - как ‘черный ящик’, характеризующийся только внешним поведением (т.н. ‘методы последовательностей’).

В качестве представителя первой группы может быть приведен язык ESTELLE (*Extended State Transition Language*), второй - язык LOTOS (*Language of Temporal Ordering Specification*); оба языка разработаны Международной организацией стандартов (ISO) и служат базовыми средствами для описания разрабатывающих международных стандартов [8].

Язык ESTELLE (1983 г.) основан на объединении логики конечного автомата (при добавлении элементов описания архитектурных особенностей протокольных систем) и языка программирования Pascal; применяемые в языке LOTOS (1984 г.) методы основаны на концепции временного упорядочения примитивов взаимодействия.

В СССР для конкретного программно-аппаратного окружения был разработан (в рамках инструментального комплекса 'Архитектор') реализующий 'автоматный метод' язык ОСА (*Описание Сетевых Архитектур*, основы и принципы языка впервые опубликованы в 1983 г.), предназначенный для реализации протокольных архитектур на вычислительных комплексах 'Эльбрус'. В комплект системы входят развитые средства анализа описаний на языке ОСА и средства тестирования и отладки (под конкретную аппаратную часть). С помощью языка ОСА были разработаны специализированные протоколы канального и сетевого уровней, транспортный и сеансовый протокол, протоколы для передачи информации и файлов, удаленного диалога и протокол удаленного запуска заданий (некоторый функциональный аналог RPC в Windows'NT).

Кроме вышеприведенных, известны системы проектирования и описания протоколов FAPL (*Format and Access Protocol Language*, 1978), PANDORA (*Protocol Analysis, Design and OpeRation Assesment*, 1982), PDIL (*Protocol Description and Implementation Language*, 1982), ПРАНАС (Каунасский политехнический институт, 1985) и др. [8].

Как и в случае традиционных языков программирования, исходный текст на языке формального описания протоколов транслируется (после этапа отладки) в машинный код, исполняемый часто (специализированными) процессорами передачи сообщений (IMP - *Interface Message Processor*).

3.4. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ АНАЛИЗА И ОПТИМИЗАЦИИ СЕТИ

В последние годы появился новый тип (сетевого) программного обеспечения, призванный обеспечивать эффективную работу сетей ЭВМ.

Дело в том, что современные компьютерные сети тяготеют к глобализации и усложнению топологии, при этом (стихийно) развивающаяся сеть часто становится неэффективной (а иногда и неработоспособной) вследствие неправильного выбора пропускных способностей и распределения потоков в сети; обычно деградация сети внешне (с точки зрения пользователя) проявля-

ется в катастрофической задержке передачи сообщений (вплоть до полной блокировки сети).

Пожалуй, впервые указанные проблемы проявились в 70-х годах в связи с постройкой и эксплуатацией сети принадлежащего Министерству обороны США Управления перспективных исследований (DARPA), принятое название - сеть ARPANET; в настоящее время данная сеть считается прообразом глобальной сети InterNet.

Сеть создавалась на случай ядерной войны и предполагала, что любой компьютер в сети может перестать функционировать в произвольный момент времени, равно как и линии связи между компьютерами. Именно такая постановка задачи привела к рождению сетевой технологии, которая впоследствие фактически стала технологией всемирной сети

ARPANET уже к 1975 году обеспечивала службу передачи сообщений между почти 100 ЭВМ, географически разнесенным по континентальной части США и подключенных спутниковой связью (через Гавайи) к нескольким точкам в Европе, причем соединенные сетью ARPANET вычислительные машины во многих отношениях являлись несовместимыми друг с другом по аппаратному и программному обеспечению. Именно тогда был обеспечен сетевой доступ к мощнейшей для того времени ЭВМ ILLIAC IV и были широко применены (с целью разгрузки вычислительных машин от выполнения задач обработки необходимых для функционирования сети сообщений) вышеупомянутые IMP.

Сеть ARPANET была спроектирована как для быстрой доставки коротких диалоговых сообщений, так и для обеспечения высокой скорости передачи длинных файлов, при этом стратегия выбора маршрута в сети принимается в каждом процессоре IMP на основе получаемой от соседних процессоров IMP информации и местной информации, включающей сведения о состоянии каналов данного IMP-процессора [9].

Заметим, что сети общего пользования сложной топологии с коммутацией пакетов появились не только в США (сети ARPANET и TELNET), но и в Канаде (DATAPAC), Англии (EPSS), Европе (EIN), Франции (TRANSPAC), Японии, Испании, Швеции и некоторых других странах.

В связи с проектированием и эксплуатацией сети ARPANET формулировались и решались задачи анализа и проектировании сетей нескольких типов. Базовой задачей является *анализ задержки* - определение средней задержки передачи сообщения по заданному пути в сети (от конкретного источника к конкретному получателю сообщений). Конечным результатом является зависимость задержки от интенсивности требований на обслуживание; обычно график этой зависимости имеет начальный малоизменяемый участок при изменении нагрузки от нуля до некоторой пороговой величины и экспоненциально возрастает при нагрузке выше пороговой (соответствующей *на-*

грузке насыщения сети), нагрузка насыщения сети соответствует блокировке сети по данному маршруту.

На рис.3.3 приведен типовой график данных расчета времени задержки, ясно видно пор**о**говое значение нагрузки сети.

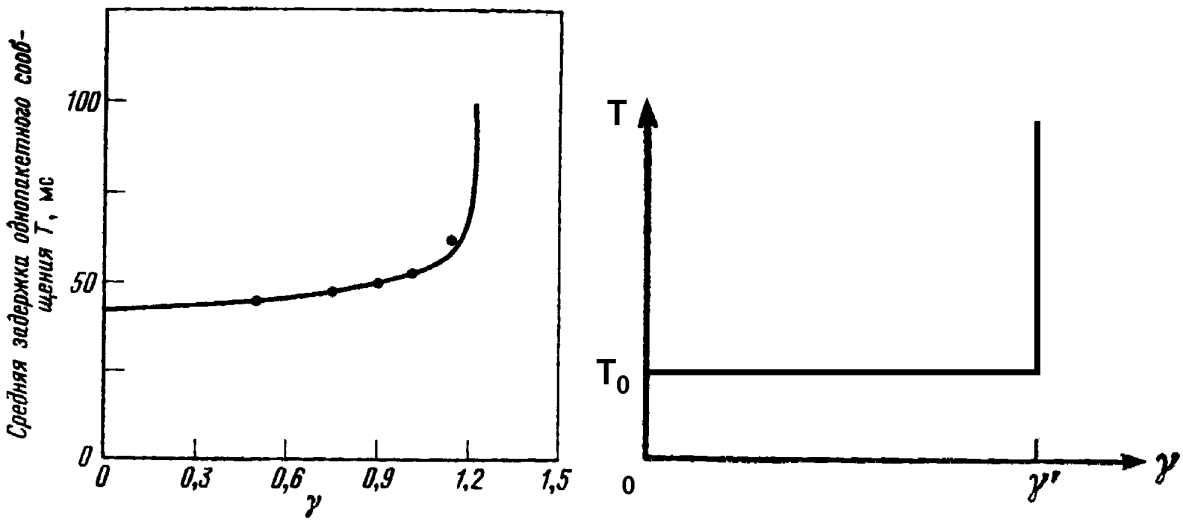

Рис.3.3. Зависимость средней времени задержки сообщения (ордината) от нагрузки в сети (абсцисса) по результатам моделирования (слева) и упрощенная пор**о**говая модель (справа).

На основе базовой задачи формулируются (более сложные) задачи расчетов и оптимизации сети:

- Задача выбора пропускных способностей (*т.н. ВПС-задача*) - оптимальный (обычно по критерию стоимости сети) выбор пропускн**ы**х способностей из *конечного набора* их возможных значений (при этом топология и потоки в сети считаются заданными).
- Задача распределения потоков (*т.н. РП-задача*) - фактически обратная вышеприведенной ВПС-задаче (заданными считаются пропускн**ы**е способности, а определяются потоки из условия минимизации средней задержки).
- Задача выбора пропускных способностей и распределения потоков - (*комбинированная ВПС/РП-задача*) - минимизация стоимости сети при заданной топологии и ограничениях на величину максимальной задержки.

При постановке задач используются несколько способов представления сети - географическая и логическая карты сети и структура сети (рис.3.1, 3.2, 3.3).

На левой части рис.3.5 показана достаточно общая структурная схема сети ЭВМ; при этом прямоугольниками представлены вычислительные средства выполнения задач обработки и хранения, соединенные друг с другом с помощью подсети связи (состоящей из коммутационных ЭВМ и высокоскоростных каналов передачи данных). Правая часть рис.3.5 иллюстрирует последний (перед этапом численного моделирования) этап представления топологии сети.

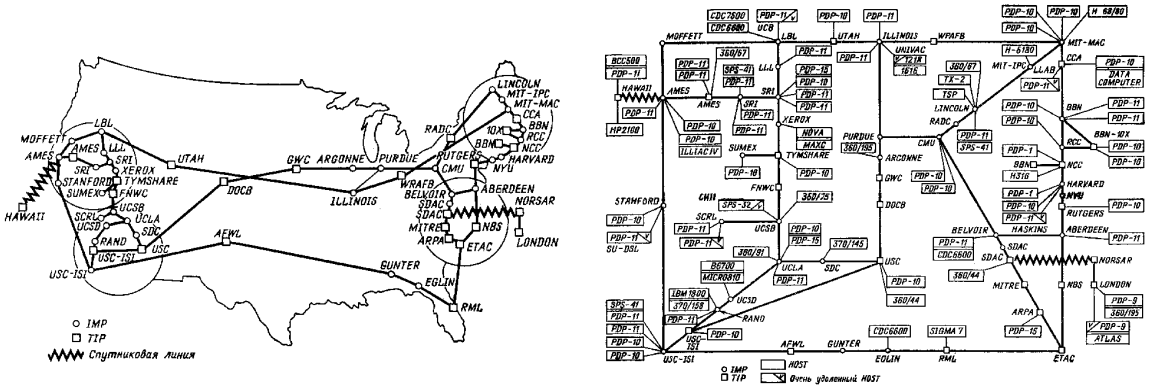

Рис.3.4. Географическая (слева) и логическая карта сети ARPANET (состояние на июнь 1975 года).

Вышеуказанные задачи трудоемки в постановке и разрешении, для получения решения необходимо применять компьютерное моделирование. При решении задач используются элементы теории графов и теории массового обслуживания, распределение потока поступления требований (на обслуживание связи) обычно принимается пуассоновским.

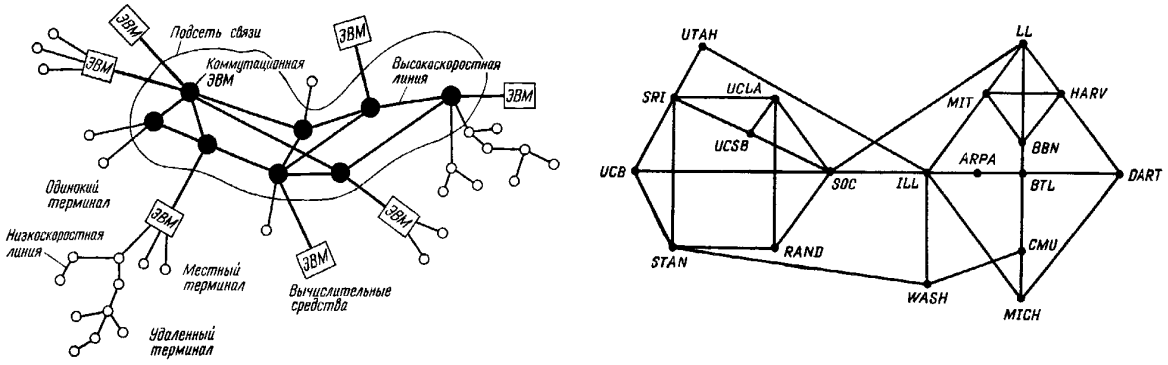

Рис.3.5. Общая структурная схема сети ЭВМ (слева) и используемая при моделировании структурная схема (справа).

Наиболее сложной задачей является задача выбора оптимальных (в соответствие с определенным критерием оптимальности - обычно стоимости)

параметров сети (в основном *маршрутов* передачи сообщений между узлами сети - при заданной топологии) при заданной максимальной средней задержке (ВТПС/РП-задача); часто используют ВМУР (*Вогнутый Метод Устранения Ребер*) и МЗР (*Метод Замены Ребер*) - алгоритмы решения задачи.

Как сформулировано в работе [9], ‘хорошая’ *процедура выбора маршрута* должна

1. Обеспечивать быструю и надежную доставку сообщений.
2. Адаптироваться к изменениям топологии сети, происходящим в результате повреждений узлов и каналов.
3. Адаптироваться к меняющейся нагрузке между парами ‘источник-получатель’.
4. Направлять пакеты в сторону от временно перегруженных узлов в сети.
5. Определять связность сети.
6. Допускать простое и автоматическое снятие и установку процессоров IMP.

Такую задачу можно решить лишь путем применения *распределенного алгоритма управления*. Это значит, что не существует центра, который принимал бы обязательные для всей сети решения, все узлы выносят *местные решения* относительно маршрутов динамическим образом. Основанное на подобных предпосылках программное обеспечение было протестировано применительно к сети ARPANET; было показано, что процедура выбора маршрутов является в основном стабильной и приводит к очень хорошим результатам, в разумной степени реагирует на повреждения узлов и каналов сети, автоматически ‘узнает’ о появлении нового узла (как только он присоединяется к сети или возвращается после исправления), эта особенность сети ARPANET является замечательной технической стороной данной сети. Для заинтересовавшихся проблемой рекомендуется работа [9]; там же приведена обширная библиография. Заметим, что в дальнейшем многие из перечисленных разработок были использованы в сети InterNet.

Таким образом, логично предположить, что в состав сетевого ПО (даже для ЭВМ уровня персонального компьютера) все чаще будут включаться решающее вышеприведенные задачи программные компоненты. Например, для обслуживания баз данных фирмой Inprise Corp. в настоящее время разрабатывается эффективная технология выбора сервера с учетом загрузки процессоров и сетевого трафика функционирующих в сети серве**о**в [7], что позволит более равномерно распределять нагрузку между серве**а**ми); в будущем они станут обязательными для системы распределенных вычислений (распределенной ОС). Представляет интерес также разработанная для сети InterNet технология (и соответствующий протокол) MPLS (*MultiProtocol Label Switching - многопроцессорная коммутация с заменой меток*), реализующая

концепции известной ATM-технологии в обобщенном виде (*Asynchronous Transfer Mode* - поддерживаемая консорциумом известных компаний технология, основанная на использовании упаковки разнородных типов данных в ячейки - ‘cells’ и создании функционирующих определенное время виртуальных соединений в физическом канале связи с целью обеспечения гарантированной по времени доставки сообщений; в настоящее время ATN-технология считается перспективной для транспортировки чувствительных к временной задержке сообщений - например, цифровой телефонии, телевидения).

4.**ИНТЕРФЕЙС СЕТЕВОЙ БАЗОВОЙ СИСТЕМЫ ВВОДА/ВЫВОДА**

Интерфейс сетевой базовой системы ввода/вывода (NetBIOS) представляет собой разработанный фирмой Microsoft Corp. интерфейс программирования, позволяющий обмениваться запросами ввода-вывода с удаленным компьютером. Пользуясь функциями NetBIOS’а, программист создает приложения, независимые от конкретной сетевой аппаратуры [11].

Расширенный пользовательский интерфейс транспортного протокола локальной сети (NetBEUI, *NetBios Extended User Interface transport*) создан фирмой IBM для работы под сетевым интерфейсом NetBIOS фирмы Microsoft Corp; протокол NetBEUI широко применялся в первых версиях Windows’NT. Несмотря на то, что этот протокол обеспечивает наивысшую скорость работы, ряд присущих ему недостатков (таких, как невозможность маршрутизации и сильная зашумленность в большой сети) позволяет эффективно использовать его только в небольших локальных сетях [5].

NetBIOS является интерфейсом сеансового уровня, м**о**гущим быть использованным приложениями для связи с NetBIOS-совместимыми транспортными протоколами (например, протокол NetBEUI). Двусторонние соединения между ЭВМ с NetBIOS реализует между ними *логическое соединение* (сеанс). После установления логического соединения компьютеры могут обмениваться данными в формате *блоков управления сетью* (NCB, *Network Control Block*) или в формате *блоков сообщений сервера* (SMB, *Server Message Block*); при настройке сетевой компоненты NetBIOS указывается *сетевое имя компьютера* (имя, под которым этот компьютер будет ‘виден’ другим пользователям сети).

Функции NetBIOS обычно не используются программистом напрямую вследствие низкого их уровня, хотя в принципе это возможно (существуют справочники по их применению, например фирменное руководство *NetBIOS programmer reference фирмы IBM Corp.*).

5.**УДАЛЕННЫЙ ВЫЗОВ ПРОЦЕДУР**

Средство *удаленного вызова процедур* (RPC, *Remote Procedure Call*) позволяет создавать приложения, состоящие из произвольного числа процедур, часть которых выполняется локально (на данном компьютере), а часть - по сети на удаленных компьютерах. Таким образом, RPC представляет модель работы с сетью, ориентированную на процедуры, а не на транспорт (передачу данных), что позволяет упростить разработку распределенных приложений.

Традиционно сетевое ПО основывается на модели ‘ввод-вывод’. В ОС Windows’NT сетевая операция начинается с того, что приложение инициирует запрос операции удаленного ввода-вывода. ОС обрабатывает запрос, передавая его редиректору (выступающему в качестве удаленной файловой системы). После обработки запроса и возврата данных удаленной файловой системой сетевая плата генерирует прерывание. Ядро ОС обрабатывает это прерывание, а исходная программа ввода-вывода возвращает результаты вызывающей программе.

PRC использует совершенно другой подход [5]. Приложения RPC структурно выглядят так же, как и обычные приложения - имеют главную программу, которая с целью выполнения специфических задач вызывает необходимые процедуры (рис.5.1).

Отличие между приложениями RPC и обычными программами состоит в том, что некоторые процедуры в приложении RPC выполняются на удаленных компьютерах, а другие - локально (рис.5.2).

Для самого приложения RPC все процедуры выглядят локальными, таким образом нет необходимости заставлять программиста писать код для передачи запроса на вычисления, ввод-вывод по сети, работы с сетевыми протоколами, обработки сетевых ошибок, ожидания результатов и т.п. - программное обеспечение RPC для Windows’NT выполняет эти задачи автоматически и для любых доступных сетевых протоколов.

Рис.5.1. Приложение, использующее библиотеки.

При проектировании приложения RPC программист должен (самостоятельно) решить, какие процедуры должны выполняться локально, а какие - удаленно. Например, при решении сводимых к операциям с матрицами большой размерности задач (типа метода конечных элементов, конечно-разностные задачи и др.) выгодно использовать мощности специальных ЭВМ с ориентированными на векторные операции процессорами (например, супер-ЭВМ серии CRAY) - если, конечно, данная рабочая станция подключена к подобной супер-ЭВМ.

Функционирует приложение RPC следующим образом. В процессе работы оно вызывает как локальные, так и отсутствующие (недоступные) на локальной машине процедуры. Для обработки последнего случая приложение связывается с *локальной DLL*, содержащей по одной *процедуре-заглушке* (*stub procedure*) для каждой из удаленной процедур. Процедура-заглушка имеет то же имя и интерфейс, что и удаленная процедура, однако вместо выполнения соответствующей операции заглушка принимает передаваемые ей параметры и выполняет операцию их *преобразования* (*marsaling*) для передачи по сети.

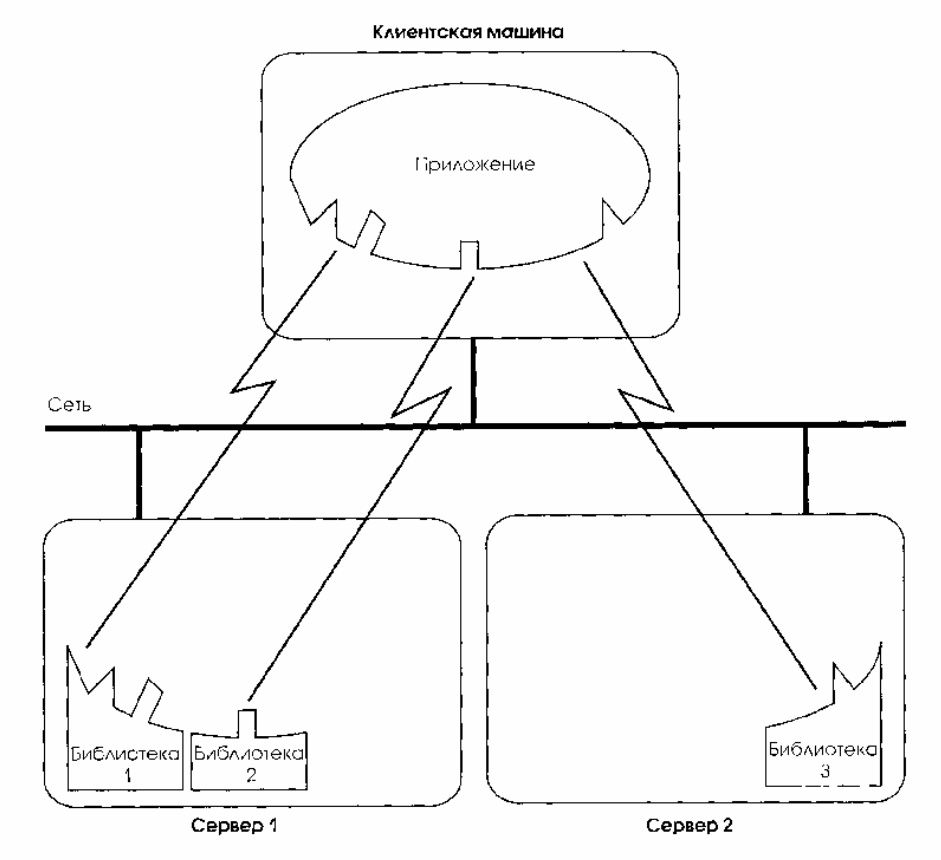

Рис.5.2. Приложение RPC, использующее библиотеки.

Под преобразованием параметров понимается их упорядочение и упаковка в определенные формат, пригодный для пересылки по сети (например, разрешение ссылок и копирование всех структур данных, на которые ссылаются указатели).

Далее заглушка вызывает процедуры библиотеки RPC *периода выполнения* (*Run Time*); они находят компьютер, на котором расположены удаленные процедуры, определяют используемые этим компьютером механизмы транспорта и посылают запрос (при помощи локального программного обеспечения сетевого транспорта). Когда удаленный компьютер (выполняющий в этот момент функцию сервера) получает запрос RPC, он выполняет обратное преобразование параметров, реконструирует оригинальный вызов процедуры и осуществляет фактический вызов ее. По окончании работы сервер выполняет обратную последовательность действий для возврата результатов вызывающей программе. На рис.5.3 схематично показано сетевое взаимодействие клиентского компьютера с серверными ЭВМ с использование RPC-библиотеки периода выполнения.

Кроме библиотеки периода выполнения, в состав средств RPC фирмы Microsoft Corp. Входит *компилятор MIDL* (*Microsoft Interface Definition Language* - язык описания интерфейса фирмы Microsoft). Использование

компилятора MIDL упрощает написание приложений RPC. Программист пишет набор обычных функций (например, на языке C или C++), описывающих удаленные процедуры, затем он добавляет к этим прототипам некоторую дополнительную информацию (например, уникальный для данной сети идентификатор пакета процедур и номер версии плюс атрибуты, указывающие, является ли параметр процедуры входом, выходом или и тем и другим). Фактически из этих модифицированных прототипов и состоит файл на *языке описания интерфейса* (IDL, *Interface Definition Language*).

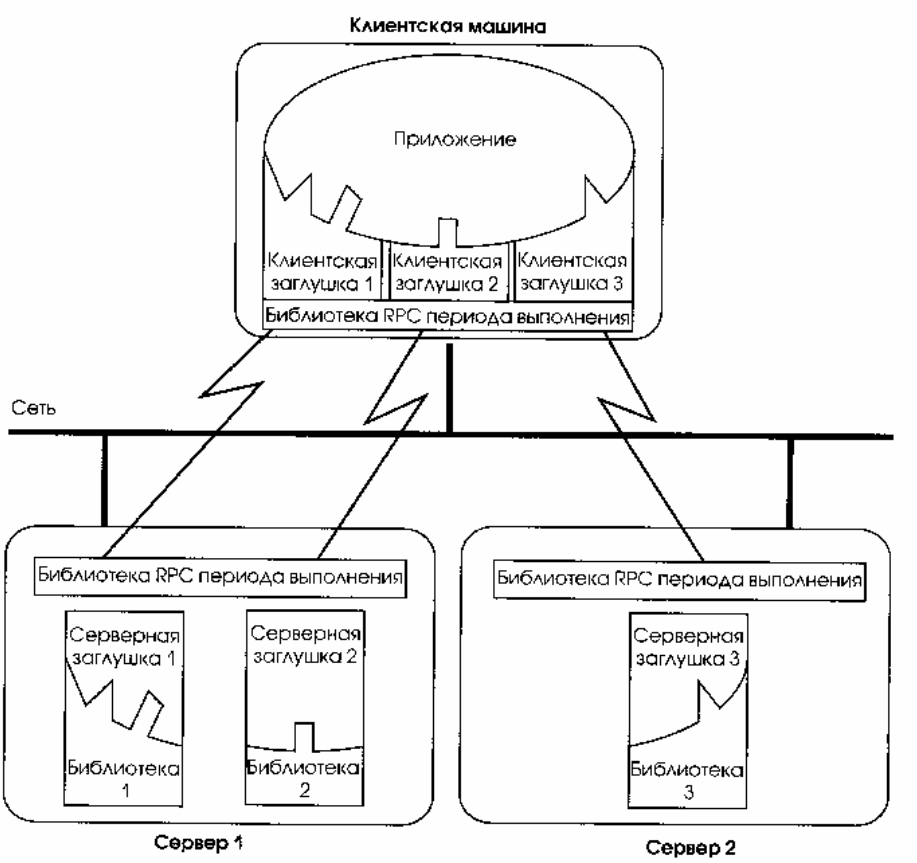

Рис.5.3. Библиотека RPC периода выполнения.

После создания файла IDL он транслируется компилятором MIDL, который и создает процедуры-заглушки для клиентской и серверной стороны, а также заголовочные файлы для подключения к приложению. При компоновке клиентского приложения совместно с файлом процедур-заглушек разрешаются все ссылки на удаленные процедуры. Используя аналогичный процесс, удаленные процедуры устанавливаются на серверной машине. Если программисту требуется только вызывать существующее приложение RPC, ему необходимо написать лишь программу для клиентской части и скомпоновать ее с локальной библиотекой RPC периода выполнения.

Библиотека RPC периода выполнения использует для взаимодействия с *транспортным протоколом* единый *интерфейс доступа к транспорту RPC* (*RPC transport interface*). Этот интерфейс служит прослойкой между средством RPC и транспортным протоколом, которая отображает операции RPC в функции, предоставляемые транспортным протоколом.

Средство RPC для Windows'NT предоставляет DLL-компоненты доступа к транспортному протоколу для именованных каналов, NetBIOS, TCP/IP и DECnet, имеется возможность разработки дополнительных DLL с целью поддержки других транспортных протоколов. Сходным образом средство RPC поддерживает работу с различными средствами защиты (при отсутствии нештатных DLL защиты программное обеспечение RPC для Windows'NT использует встроенную защиту именованных каналов).

Для обеспечения взаимодействия средства RPC с приложениями RPC на другой ЭВМ они должны использовать одинаковые соглашения RPC. Microsoft RPC соответствует стандарту RPC, установленному *Open Software Foundation* (OSF) в спецификации *среды распределенных вычислений* (DCE, *Distribute Calculation Environment*). Таким образом, написанные с использованием Microsoft RPC приложения могут вызывать удаленные процедуры на других системах, использующих стандарт DCE.

Большинство сетевых сервисов Windows'NT являются приложениями RPC и поэтому могут вызываться как локальными процессами, так и процессами на удаленных машинах. Это значит, что удаленный компьютер может обращаться к сервисам данной ЭВМ для просмотра совместно используемых ресурсов, открытых файлов, очередей печати или активных пользователей на этом сервере, либо он может вызвать сервис сообщений для посылки сообщений (при наличии соответствующих прав д**о**ступа).

Существуют более совершенные механизмы реализации вызова удаленных процедур - *асинхронные вызовы удаленных функций* (ARPC, *Asynchronous Remote Procedure Cal*), позволяющие на основе применения *функций отклика* (*call-back function*) избежать приостановки выполнения прикладной программы на локальной машине. Для связи с удаленными системами RPC может использовать сервис NetBIOS, Windows Sockets и другие доступные средства (например, именованные каналы для Windows'NT). Для вызова процедур, расположенных на том же компьютере, что и вызывающая программа, и обмена информацией с ними служит механизмы *вызова локальных процедур* (LPC, *Local Procedure Call*) или *упрощенного вызова удаленных процедур* (LRPC, *Lightweight Remote Procedure Call*). Однако вышеуказанные средства доступны лишь в Windows'NT.

Резюмируя сказанное, следует признать, что предложенная Microsoft RPC-методика, значительно упрощая создание сетевых приложений (путем сокрытия от программиста многих рутинных действий), все же весьма далека от совершенства. В самом деле, для изменения многих параметров сети

приходится вновь перекомпилировать приложение, изменяя должным образом параметры IDL. На самом деле RPC-подход, конечно, является компромиссом между современным уровнем (не очень хорошо совместимых между собой) локальных операционных систем и (нереализованной пока) *распределенной ОС*.

Активно продвигаемая в настоящее время технология dotNet фирмы Microsoft Corp. позволяет просто и единообразно (невзирая на используемый язык программирования) создавать сетевые приложения и контролировать их исполнение (подробнее см. раздел 7.2.2).

6.СОКЕТЫ, ДАТАГРАММЫ И КАНАЛЫ СВЯЗИ

В локальных и глобальных сетях существует два принципиально разных способа передачи данных - *датаграммный* и *потоковый* [13].

Первый из них предполагает *посылку пакетов данных от одного узла другому (или сразу нескольким узлам)* без получения подтверждения о доставке и даже без гарантии того, что передаваемые пакеты будут получены в правильной последовательности. Примером такого протокола может служить протокол UDP (*User Datagram Protocol*), который используется в сетях TCP/IP, или протокол IPX, который является базовым в сетях Novell NetWare.

Основные преимущества датаграммных протоколов заключаются в высоком быстродействии и возможности широковещательной передачи данных, когда один узел отправляет сообщения, а другие их получают, причем все одновременно.

Второй способ передачи данных предполагает *создание канала передачи данных* между двумя различными узлами сети. При этом канал создается средствами датаграммных протоколов, однако доставка пакетов в канале является гарантированной. Пакеты всегда доходят в целостности и сохранности, причем в правильном порядке, хотя быстродействие получается в среднем ниже за счет посылки подтверждений. Примерами протоколов, использующих каналы связи, могут служить протоколы TCP и SPX (протокол NetBIOS допускает передачу данных с использованием как датаграмм, так и каналов связи).

Для передачи данных с использованием любого из перечисленных выше способов каждое приложение должно создать объект, который называется *сокетом*. 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). Сокет обеспечивает конечную точку соединения и функционирует как двунаправленный канал для входящих и исходящих данных между компьютерами в сети.

При программировании сокет более всего похож на идентификатор файла (*file handle*), который нужен для выполнения над файлом операций чтения или записи. Прежде чем приложение, запущенное на узле сети, сможет выполнять передачу или прием данных, оно должно создать сокет и проинициализировать его, указав некоторые параметры.

Работа с именованными и анонимными каналы передачи данных и каналами типа Mailslot подробно (включая исходные тексты приложений на C++) описана в работе [14].

6.1. ИНИЦИАЛИЗАЦИЯ ПРИЛОЖЕНИЯ И ЗАВЕРШЕНИЕ ЕГО РАБОТЫ

В процессе инициализации приложение должно зарегистрировать себя в библиотеке **WSOCK32.DLL**, которая предоставляет приложениям интерфейс **Windows Sockets** в среде операционных систем Microsoft Windows'9x и Windows'NT.

Для инициализации необходимо вызвать функцию **WSAStartup**, прототип которой имеет вид:

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

В параметре **wVersionRequested** указывается версия интерфейса **Windows Sockets**, необходимая для работы приложения. Старший байт параметра указывает младший номер версии (minor version), младший байт - старший номер версии (major version).

Перед вызовом функции **WSAStartup** параметр **lpWSAData** должен содержать указатель на структуру типа **WSADATA**, в которую будут записаны сведения о конкретной реализации интерфейса **Windows Sockets**.

В случае успеха функция **WSAStartup** возвращает нулевое значение. Если происходит ошибка, возвращается одно из следующих значений

Значение	*Описание*
WSASYSNOTREADY	Сетевое программное обеспечение не готово для работы
WSAVERNOTSUPPORTED	Функция не поддерживается данной реализацией интерфейса **Windows Sockets**
WSAAEINVAL	DLL-библиотека, обеспечивающая интерфейс **Windows Sockets**, не соответствует версии, указанной приложением в параметре **wWersionRequested**

Ниже представлен фрагмент исходного текста описанного в [13] приложения **SERVER**, выполняющий инициализацию интерфейса **Windows Sockets:**

```
rc = WSAStartup(MAKEWORD(1,1),  &WSAData);

if (rc)
{
   MessageBox(NULL, "Ошибка вызова WSAStartup",  "Error", MB_OK);
   return (FALSE);
}

// Отображение описания и версии системы Windows
// Sockets в окне органа управления StatusBar
wsprint(szTemp, "Сервер использует %s %s",
                    WSAAData.szDescription,
                    WSAData.szSystemStatus);

// Создать StatusBar и реально отобразить  строку szTemp
hwndSd = CreateStatusWindow(WS_CHILD | WS_VISIBLE |
                                    WS_BORDER | SBARS_SIZEGRIP,
                                    szTemp,  hWnd,  IDS_STATUSBAR);
```

В операционных системах Microsoft Windows'9x и Windows'NT версии 3.51 встроена система **Windows Sockets** версии 1.1, поэтому именно это значение указано при вызове функции **WSAStartup**.

Вышеприведенный код должен быть дополнен описанием структуры **WSADATA** и указателя **LPWSADATA** на нее:

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

typedef   WSAData  FAR  *LPWSADATA;
```

Вышеиспользованные поля **szDescription** и **szSystemStatus** после вызова функции **WSAStartup** содержат соответственно описание конкретной реализации интерфейса **Windows Socket** и текущее состояние этого интерфейса в виде текстовых строк.

В полях **wVersion** и **wHighVersion** содержатся соответственно версия спецификации **Windows Socket**, которую будет использовать приложение, и

версия спецификации, которой соответствует конкретная реализация интерфейса **Windows Socket**.

Приложение может одновременно создать несколько сокетов, например, для использования в разных подзадачах одного процесса. В поле **iMaxSockets** хранится максимальное количество сокетов, которое можно получить для одного процесса. В поле **iMaxUdpDg** записан максимальный размер пакета данных, который можно переслать с использованием датаграммного протокола **UDP**; поле **lpVendorInfo** содержит указатель на дополнительную информацию, формат которой зависит от фирмы-изготовителя конкретной реализации системы **Windows Sockets**.

Перед тем как завершить свою работу, приложение должно освободить ресурсы, полученные у операционной системы для работы с **Windows Sockets**. Для выполнения этой задачи приложение должно вызвать функцию **WSACleanup**, определенную следующим образом:

```
int
WSACleanup(void);
```

Эта функция может возвратить нулевое значение при успехе или значение **SOCKET_ERROR** в случае ошибки.

Для получения кода ошибки следует воспользоваться функцией с именем **WSAGetLastError:**

```
int
WSAGetLastError(void);
```

Функция **WSAGetLastError** позволяет определить код ошибки при неудачном завершении практически всех функций интерфейса **Windows Socket**. Ее следует вызывать сразу вслед за функцией, завершившейся неудачно.

Если ошибка возникла при выполнении функции **WSACleanup**, функция **WSAGetLastError** возращает одно из следующих значений

Значение	*Описание*
WSANOTINITIALISED	Интерфейс **Windows Sockets** не был проинициализирован функцией **WSAStartup**
WSAENETDOWN	Сбой сетевого программного обеспечения
WSAEINPROGRESS	Во время вызова функции **WSACleanup** выполнялась одна из блокирующих функций интерфейса **Windows Sockets**

Представляет интерес случай возврата кода ошибки **WSAEINPROGRESS**.

Некоторые функции интерфейса **Windows Sockets** способны блокировать работу приложения, т.к. не возвращают управления вызывающей программе до своего завершения. В использующих вытесняющую многозадачность ОС (таких, как Windows'95x и Windows'NT) это не приводит к блокировке всей системы, однако можно избежать блокирующих функций путем использования предоставляемых **Windows Sockets** асинхронных аналогов этих функций.

6.2.СОЗДАНИЕ И ИНИЦИАЛИЗАЦИЯ СОКЕТА

После инициализации интерфейса **Windows Sockets** приложение должно создать один или несколько сокетов, которые будут использованы для передачи данных.

Сокет создается с помощью функции **socket**, имеющей следующий прототип:

```
SOCKET
socket(int af, int type, int protocol);
```

Параметр **af** определяет формат адреса. Для этого параметра следует указывать значение **AF_INET**, что соответствует формату адреса, принятому в InterNet. Параметры **type** и **protocol** определяют соответственно тип сокета и протокол, который будет использован для данного сокета, можно указывать сокеты следующих двух типов

Тип сокета	*Описание*
SOCK_STREAM	Сокет будет использован для передачи данных через канал связи с использование протокола **TCP**
SOCK_DGRAM	Передача данных будет выполняться без создания каналов связи через датаграммный протокол **UDP**

Что же касается параметра **protocol**, то для него следует указать нулевое значение.

В случае успеха функция **socket** возвращает дескриптор (тип **SOCKET**), который следует использовать для выполнения всех операций над данным сокетом; имеется прямая аналогия между дескрипторами файла и сокета, однако над последним не определены некоторые (физически недопустимые для сокета) операции (например, позиционирование указателя в потоке). Если же произошла ошибка, функция **socket** возвращает значение **INVALID_SOCKET**. Для анализа причины ошибки следует вызвать функцию **WSAGetLastError**, которая в данном случае может вернуть один из следующих кодов ошибки

Код ошибки	*Описание*
WSANOTINITIALISED	Интерфейс **Windows Sockets** не был проинициализирован функцией **WSAStartup**
WSAENETDOWD	Сбой сетевого программного обеспечения
WSAEAFNOSUPPORT	Указан неправильный тип адреса
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса **Windows Sockets**
WSAEMFILE	Израсходован весь запас свободных дескрипторов
WSAENOBUFS	Нет памяти для создания буферов
WSAEPROTONOSUPPORT	Указан неправильный протокол
WSAEPROTOTYPE	Указанный протокол несовместим с данным типом сокета
WSAESOCKNOSUPPORT	Указанный тип сокета несовместим с данным типом адреса

Ниже приведен фрагмент кода, в котором создается сокет для передачи данных с использование протокола TCP

```
srv_socket = socket(AF_INET, SOCK_STREAM, 0);

if (srv_socket == INVALID_SOCKET)
{
   MessageBox(NULL, "Ошибка создания сокета", "Error", MB_OK);
   return;
}
```

При практическом программировании после выявления ошибки выполнения функции следует сразу же вызвать функцию **WSAGetLastError** и некоторым образом информировать пользователя о конкретной причине ошибки.

6.3.**УДАЛЕНИЕ СОКЕТА**

Для освобождения ресурсов приложение должно закрывать сокеты, которые ему больше не нужны, вызывая функцию **closesocket**

```
int
closesocket(SOCKET sock);
```

Ниже перечислены коды ошибок для этой функции

Код ошибки	*Описание*
WSANOTINITIALISED	Перед использование функции **closesocket** не была вызвана функция **WSAStartup**
WSAENETDOWN	Сбой в сети

WSANOTSOCK	Указанный в параметре дескриптор не является дескриптором сокета
WSAINPROGRESS	Выполняется блокирующая функция интерфейса **Windows Sockets**
WSAEINTR	Работа функции была отменена при помощи функции **WSACancelBlockingCall**

6.4.**ПАРАМЕТРЫ СОКЕТА**

Перед использованием сокета необходимо задать его параметры, жля чего следует подготовить структуру типа **sockaddr**, определение которой помещено ниже

```
struct  sockaddr
{
   u_short  sa_family;
   char       sa_data[14];
} ;

typedef  struct sockaddr         SOCKADDR;
typedef  struct sockaddr        *PSOCKADDR;
typedef  struct sockaddr  FAR *LPSOCKADDR;
```

Для работы с адресами в формате InterNet используется другой вариант этой структуры, в котором детализируется формат поля **sa_data**

```
struct  sockaddr_in
{
   short     sin_family;
   u_short  sin_port;
   struct    in_addr  sin_addr;
   char       sin_zero[8];
} ;

typedef  struct  sockaddr_in          SOCKADDR_IN;
typedef  struct  sockaddr_in         *PSOCKADDR_IN;
typedef  struct  sockaddr_in  FAR  *LPSOCKADDR_IN;
```

Поле **sin_family** определяет тип адреса. Следует записать в это поле значение **AF_INET**, которое соответствует типу адреса, принятому в InterNet (структура **srv_address** имеет тип **SOCKADDR_IN**)

```
srv_address.sin_family = AF_INET;
```

Поле **sin_port** определяет номер порта, который будет использоваться для передачи данных. Порт - это просто идентификатор программы, выпол-

няющей обмен по сети. На одном узле может одновременно работать несколько программ, использующих разные порты.

Особенностью поля **sin_port** является использование так называемого *сетевого формата данных*. Этот формат отличается от того, что принят в процессорах с архитектурой **Intel**, а именно - младшие байты данных хранятся по старшим адресам памяти (архитектура процессоров **Intel** подразумевает хранение старших байтов данных по младшим адресам).

Сетевой формат данных удобен при организации глобальных сетей, так как в узлах такой сети могут использоваться компьютеры с различной архитектурой.

Для выполнения преобразований из обычного формата в сетевой и обратно в интерфейсе **Windows Socket** предусмотрен специальный набор функций. В частности, для заполнения поля **sin_port** нужно использовать функцию **htons**, выполняющую преобразование 16-разрядных данных из формата **Intel** в сетевой формат.

Ниже показано, как инициализируется поле **sin_port** в приложении **SERVER**, полностью описанном в [13]

```
#define  SERV_PORT  5000

srv_address.sin_port = htons(SERV_PORT);
```

Поле **sin_addr** структуры **sockaddr_in** представляет собой структуру **in_addr**

```
struct  in_addr
{
   union
   {
      struct  { u_char s_b1,  s_b2,  s_b3,  s_b4;  }  S_un_b;
      struct  { u_short  s_w1,  s_w2; }  S_un_w;
      u_long  S_addr;
      S_un;
   }
} ;

#define  s_addr    S_un.S_addr;
#define  s_host    S_un.S_un_b.s_b2;
#define  s_net     S_un.S_un_b.s_b1;
#define  s_imp     S_un.S_un_w.s_w2;
#define  s_impno  S_un.S_un_b.s_b4;
#define  s_lh       S_un.S_un_b.s_b3;
```

При инициализации сокета в этой структуре следует указать адрес **IP** (32-битовое уникальное число, идентифицирующее данный компьютер, о

принципах IP-адресации подробнее см. подраздел 7.1.1), с которым будет работать данный сокет. Если сокет будет работать с любым адресом (например, создается сервер, который будет доступен из узлов с любым адресом). Адрес для сокета может быть указан следующим образом

```
srv_address.sin_addr.s_addr = INADDR_ANY;
```

В том случае, если сокет будет работать с определенным IP-адресом (например, создается приложение-клиент, которое будет обращаться к серверу с конкретным адресом IP), в указанную структуру необходимо записать реальный IP-адрес.

Датаграммный протокол **UDP** позволяет посылать пакеты данных одновременно всем рабочим станциям в широковещательном режиме. Для этого необходимо указать адрес как **INADDR_BROADCAST**.

Если известен адрес в виде четырех десятичных чисел, разделенных точкой (именно так его может вводить пользователь), то можно заполнить поле адреса при помощи функции **inet_addr** (структура **dest_sin** имеет тип **SOCKADDR_IN**)

```
dest_sin.sin_addr.s_addr = inet_addr("200.200.200.201");
```

В случае ошибки функция возвращает значение **INADDR_NONE**, что и следует использовать для проверки.

Обратное преобразование адреса IP в текстовую строку можно при необходимости легко выполнить с помощью функции **inet_ntoa**, имеющий следующий прототип

```
char FAR *
inet_ntoa(struct in_addr in);
```

При ошибке эта функция возвращает **NULL**.

Однако чаще всего пользователь работает с доменными именами, применяя сервер DNS или файл HOSTS (см. подраздел 7.1.1). В этом случае следует воспользоваться функцией **gethostbyname**, возвращающей адрес IP, а затем записать полученный адрес в структуру **sin_addr**

```
PHOSTENT phe;

phe = gethostbyname(“ftp.microsoft.com”);

if (phe == NULL)
{
   closesocket(srv_socket);
   MessageBox(NULL, "Ошибка функции GetHostByName",
```

```
                        "Error",  MB_OK);
   return;
}

memcpy((char FAR *) & (dest_sin.sin_addr), phe->h_addr, phe->h_length);
```

В случае ошибки функция **gethostbyname** возвращает **NULL**, после чего причину ошибку можно выяснить путем проверки кода возврата функцией **WSAGetLastError**.

Если указанный адрес найден в базе DNS или файле HOSTS (см. подраздел 7.1.1), функция **gethostbyname** возвращает указатель на структуру **hostent**, описанную ниже

```
struct  hostent
{
   char  FAR  *h_name;              // имя узла
   char  FAR *FAR  *h_aliases;      // список альтернативных имен
   short  h_addrtype:               // тип адреса узла
   short  h_lenght;                 // длина адреса
   char  FAR  *FAR  *h_addr_list;   // список адресов
#define h_add r  h_addr_list[0];    // адрес
};

typedef  struct  hostent       *PHOSTENT;
typedef  struct  hostent  FAR *LPHOSTENT;
```

Искомый адрес находится в первом элементе списка **h_addr_list[0]**, на который можно также ссылаться при помощи **h_addr**, длина поля адреса находится в поле **h_length**.

6.5.ПРИВЯЗКА АДРЕСА К СОКЕТУ

После подготовки структуры **SOCKADDR** (записи в нее параметров сокета - в частности, адреса) следует привязку адреса к сокету при помощи функции **bind**

```
int
bind(SOCKET sock, const struct sockaddr FAR *addr,   int namelen);
```

Параметр **sock** содержит дескриптор созданного ранее функцией **socket** сокета, в поле **addr** следует записать указатель на подготовленную структуру **SOCKADDR**, в поле **namelen** - размер этой структуры.

В случае ошибки функция **bind** возвращает значение **SOCKET_ERROR**, дальнейший анализ причин ошибок следует выполнять

при помощи функции **WSAGetLastError**, возможные коды ошибок перечислены ниже

Код ошибки	*Описание*
WSANOTINITIALISED	Перед использование функции необходимо вызвать функцию **WSAStartup**
WSAENETDOWN	Сбой в сети
WSAEADDRINUSE	Указанный адрес уже используется
WSAEFAULT	Значение параметра **namelen** меньше размера структуры **sockaddr**
WSAINPROGRESS	Выполняется блокирующая функция интерфейса **Windows Sockets**
WSAEAFNOSUPPORT	Выбранный протокол не может работать с указанным семейством адресов
WSAEINVAL	Сокет уже привязан к адресу
WSAENOBUFS	Установлено слишком много соединений
WSAENOTSOCK	Указанный в параметре дескриптор не является дескриптором сокета

Пример вызова функции **bind** показан ниже

```
if (bind(srv_socket, (LPSOCKADDR) &srv_address,
                    sizeof(srv_address)) == SOCKET_ERROR)
{
   closesocket(srv_socket);
   MessageBox(NULL, "Ошибка функции Bind",  "Error",  MB_OK);
   return;
}
```

6.6.**СОЗДАНИЕ КАНАЛА СВЯЗИ**

В случае передачи датаграммных сообщений при помощи протокола негарантированной доставки **UDP** канал связи не нужен и сразу после создания сокетов и их инициализации можно приступать к передаче данных. Однако при использовании протокола гарантированной доставки **TCP** необходимо создать канал связи.

6.6.1.**СТОРОНА СЕРВЕРА**

При создании канала связи со стороны сервера прежде всего следует переключить сокет в режим приема для выполнения ожидания соединения с клиентом при помощи функции **listen**

```
int
listen(SOCKET sock, int backlog);
```

Через параметр **sock** функции необходимо передать дескриптор сокета, который будет использован для создания сокета, параметр **backlog** задает максимальный размер очереди для ожидания соединения (можно указывать значения от 1 до 5). Очередь содержит запросы на установку соединений для каждой пары значений (IP-адрес, порт). Ниже приведен список возможных кодов ошибок для функции **listen**

Код ошибки	*Описание*
WSANOTINITIALISED	Перед использованием функции необходимо вызвать функцию **WSAStartup**
WSAENETDOWN	Сбой в сети
WSAEADDRINUSE	Указанный адрес уже используется
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса **Windows Sockets**
WSAEINVAL	Сокет еще не был привязан к адресу или уже находится в подключенном состоянии
WSAEISCONN	Сокет уже находится в подключенном состоянии
WSAEMFILE	Нет доступных дескрипторов
WSAENOBUFS	Недостаток памяти для размещения буфера
WSAENOTSOCK	Указанный в параметре дескриптор не является дескриптором сокета
WSAEOPNOTSUPP	Функция **listen** не может работать с сокетом указанного типа

Ниже приведен пример вызова функции **listen**

```
if (listen(srv_socket, 1) == SOCKET_ERROR)
{
   closesocket(srv_socket);
   MessageBox(NULL, "Ошибка функции Listen",  "Error",  MB_OK);
   return;
}
```

Далее программа должна ожидать соединения, это может быть выполнено двумя различными способами.

Первый способ заключается в циклическом вызове функции **accept** до тех пор, пока не будет установлено соединение; затем можно приступить к обмену данными.

Функция **accept** имеет следующий прототип

```
SOCKET
accept(SOCKET sock, struct sockaddr FAR *addr,  int FAR *addrlen);
```

Через параметр **sock** передается сокета, находящегося в режиме приема для выполнения ожидания. Параметр **addr** должен содержать адрес буфера, в который будет записан адрес узла, подключившегося к серверу (размер этого буфера должен быть записан в целочисленной переменной, адрес которой передается через параметр **addrlen**).

Список возможных кодов ошибки для функции **accept** приведен ниже

Код ошибки	*Описание*
WSANOTINITIALISED	Перед использованием функции необходимо вызвать функцию **WSAStartup**
WSAENETDOWN	Сбой в сети
WSAEFAULT	Значение параметра **addrlen** меньше размера структуры адреса
WSAEINTR	Выполнение функции было отменено при помощи функции **WSACancelBlockingCall**
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса **Windows Sockets**
WSAEINVAL	Перед вызовом функции **accept** не была вызвана функция **listen**
WSAEMFILE	Нет доступных дескрипторов
WSAENOBUFS	Установлено слишком много соединений
WSAENOTSOCK	Указанный в параметре дескриптор не является дескриптором сокета
WSAEOPNOTSUPP	Данный тип сокета нельзя использовать при вызове функций, ориентированных на работу с каналом связи
WSAEWOULDBLOCK	Сокет отмечен как неблокирующий и в настоящее время нет каналов связи, которые нужно устанавливать

Второй (более удачный) способ ожидания соединения заключается в использовании расширения программного интерфейса **Windows Sockets**, предназначенного для выполнения *асинхронных операций*.

Вместо того, чтобы ожидать соединения, вызывая в цикле функцию **accept**, приложение может единожды вызвать функцию **WSAAsyncSelect**, указав ей, что при получении запроса на установку соединения оконная функция приложения должна получить сообщение

```
#define  WSA_ACCEPT  (WM_USER+1)

// При попытке установки соединения главное окно
// приложения получит сообщение WSA_ACCEPT

rc = WSAAsyncSelect(srv_socket,  hWnd,                    WSA_ACCEPT,
                    FD_ACCEPT);

if (rc > 0)
{
```

```
   closesocket(srv_socket);
   MessageBox(NULL, "Ошибка функции WSAAsyncSelect",
                        "Error",  MB_OK);
   return;
}
```

В данном случае ожидание выполняется для сокета **srv_socket**. Последний параметр функции имеет параметр **FD_ACCEPT**, это означает, что при попытке создания канала связи функция окна с дескриптором **hWnd** получит сообщение **WSA_ACCEPT**, определенное в приложении.

Обработчик этого сообщения может выглядеть, например, следующим образом

```
void
WndProc_OnWSAAccept(HWND hWnd,
                            UINT msg,
                            WPARAM wParam,
                            LPARAM lParam);
{
   int rc;

   // при ошибке отменяется поступление извещений
   // в главное окно приложения
   if (WSAGETSELECTERROR(lParam) != 0)
   {
      MessageBox(NULL, "Ошибка функции Accept",  "Error",  MB_OK);
      WSAAsyncSelect(srv_socket, hWnd, 0, 0);
      return;
   }

   // определение размера сокета
   acc_sin_len = sizeof(acc_sin);

   // разрешение установки соединения
   srv_socket = accept(srv_socket, (LPSOCKADDR) & acc_sin,
                        (int FAR *) & acc_sin_len);

   if (srv_socket == INVALID_SOCKET)
   {
      MessageBox(NULL, "Ошибка функции Accept, недопустимый сокет",
                           "Error", MB_OK);
      return;
   }

   // если на данном сокете начнется передача данных от
   // клиента, в главное окно приложения поступит
   // сообщение WSA_NETEVENT.
   // Это же сообщение поступит при разрыве соединения
```

```
   rc = WSAAsyncSelect(srv_socket, hWnd, WSA_NETEVENT,
                              FD_READ | FD_CLOSE);

   if (rc > 0)
   {
      closesocket(srv_socket);
      MessageBox(NULL, "Ошибка функции WSAAsyncSelect",
                       "Error",  MB_OK);
      return;
   }
} // конец функции WndProc_OnWSAAccept
```

В данном случае обработчик сообщения вначале вызывает функцию **accept**, выполняющую создание канала передачи данных. После этого функция **WSAAsyncSelect** вызывается повторно для того, чтобы установить асинхронную обработку приема данных от удаленного клиента, а также обработку ситуации разрыва канала связи.

6.6.2.СТОРОНА КЛИЕНТА

Рассмотрим процедуру установки канала связи со стороны клиента, использованную в приложении **CLIENT**, полностью приведенном в [13].

Для установки соединения в приложении используется функция **SetConnected**

```
SOCKADDR_IN  dest_sin;

void SetConnection(HWND hWnd)
{
   PHOSTENT  phe;

   // создание сокета
   srv_socket = socket(AF_INET, SOCK_STREAM, 0);

   if (srv_socket == INVALID_SOCKET)
   {
      MessageBox(NULL, "Ошибка создания сокета",  "Error",  MB_OK);
      return;
   }

   // установка IP-адреса и номера порта
   dest_sin.sin_family = AF_INET;

   // определение адреса узла
   phe = gethostbyname("localhost");
```

```
   if (phe == NULL)
   {
      closesocket(srv_socket);
      MessageBox(NULL, "Ошибка функции GetHostByName",
                       "Error",  MB_OK);
      return;
   }

   // копирование адреса узла
   memcpy((char FAR *) &(dest_sin.sin_addr), phe->h_addr, phe->h_lenght);

   // копирование номера порта
   dest_sin.sin_port = htons(SERV_PORT);

   // установка соединения
   if (connect(srv_socket, (PSOCKADDR) &dest_sin, sizeof(dest_sin)) > 0)
   {
      closesocket(srv_socket);
      MessageBox(NULL, "Ошибка соединения",  "Error",  MB_OK);
      return;
   }
} // конец функции SetConnection
```

В вышеприведенном листинге вначале с помощью функции **socket** создается сокет, затем выполняется заполнение адресной информацией структуры **dest_sin**. С целью получения адреса IP используется функция **gethostbyname**, в качестве параметра указывается имя узла **localhost**. В файле **HOSTS** это имя всегда отображается на адрес 127.0.0.1

```
. . . другие строки файла localhost . . .
127.0.0.1  localhost
. . . другие строки файла localhost . . .
```

Адрес 127.0.0.1 является локальным, его можно использовать для тестирования приложений, выполняющих обмен данных при помощи протокола TCP/IP, запуская серверное и клиентское приложения на одном компьютере.

После заполнения структуры с адресной информацией функция **connect** создает канал связи с сервером.

6.7.ПЕРЕДАЧА И ПРИЕМ ДАННЫХ

После создания канала связи можно начинать передачу данных. Для передачи данных посредством протокола гарантированной доставки TCP можно воспользоваться функциями **send** и **recv**, входящими в программный интерфейс **Windows Sockets**.

Функция передачи данных **send** имеет 4 параметра - дескриптор сокета **sock**, на котором выполняется передача, адрес буфера **buf**, содержащего передаваемое сообщение, размер этого буфера **bufsize** и флаги **flags**

```
int send(SOCKET sock, const char FAR *buf, int bufsize, int flags);
```

Фрагмент исходного С-кода передачи данных серверу приведен ниже

```
char  szBuf[80];
lstrcpy(szBuf, "Тестовая строка для передачи");
send(srv_socket, szBuf, lstrlen(szBuf), 0);
```

Параметры предназначенной для приема данных функции **recv** аналогичны параметрам функции **send**

```
int
recv(SOCKET sock, char FAR *buf, int bufsize, int flags);
```

Функции **send** и **recv** возвращают количество соответственно принятых и переданных байтов данных. Принимающее данные приложение должно вызывать функцию **recv** в цикле до тех пор, пока не будут приняты все переданные данные, при этом на один вызов функции **send** может приходиться несколько вызовов функции **recv**.

В случае ошибки обе эти функции возвращают значение **SOCKET_ERROR**, для тщательного анализа ошибки следует использовать функцию **WSAGetLastError**.

В нижерасположенной таблице приведен список кодов ошибок, которые могут возникать при вызове функции **send**

Код ошибки	*Описание*
WSANOTINITIALISED	Перед использованием функции необходимо вызвать функцию **WSAStartup**
WSAENETDOWN	Сбой в сети
WSAEACCES	Указанный адрес является широковещательным (**broadcast**), однако перед вызовом функции не был установлен соответствующий флаг
WSAEINTR	Выполнение функции было отменено при помощи функции **WSACancelBlockingCall**
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса **Windows Sockets**
WSAEFAULT	Параметр **buf** указан некорректно (он не указывает на адресное пространство, принадлежащее приложению)
WSAENETRESET	Необходимо сбросить соединение
WSAENOBUFS	Возникла блокировка буфера
WSAENOTCONN	Сокет не подсоединен

WSAENOTSOCK	Указанный в параметре дескриптор не указывает на сокет
WSAESHUTDOWN	Сокет не был закрыт функцией **shutdown**
WSAEWOULDBLOCK	Сокет отмечен как неблокирующий, но запрошенная операция приведет к блокировке
WSAEMSGSIZE	Был использован сокет типа **SOCK_DGRAM** (предназначенный для передачи датаграмм), но при этом размер пакета данных превышает максимально допустимый для данной реализации интерфейса **Windows Sockets**
WSAEINVAL	Сокет не был подключен функцией **bind**
WSAECONNABORTED	Сбой соединения из-за слишком большой задержки или по другой причине
WSAECONNRESET	Разрыв соединения удаленным узлом сети

При выполнении функции **recv** могут возникнуть следующие ошибки

Код ошибки	*Описание*
WSANOTINITIALISED	Перед использованием функции необходимо вызвать функцию **WSAStartup**
WSAENETDOWN	Сбой в сети
WSAENOTCONN	Сокет не был подсоединен
WSAEINTR	Выполнение функции было отменено при помощи функции **WSACancelBlockingCall**
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса **Windows Sockets**
WSAENOTSOCK	Указанный в параметре дескриптор не есть дескриптор сокета
WSAESHUTDOWN	Сокет был закрыт функцией **shutdown**
WSAEWOULDBLOCK	Сокет отмечен как неблокирующий, но запрошенная операция приведет к блокировке
WSAEMGSIZE	Размер пакета данных превышает размер буфера, в результате чего принятый пакет был обрезан
WSAEINVAL	Сокет не был подключен функцией **bind**
WSAECONNABORTED	Сбой из-за слишком большой задержки или по другой причине
WSAECONNRESET	Разрыв соединения удаленным узлом

Передача и прием данных в цикле может привести к блокировке работы приложения. Если это неприемлемо, следует воспользоваться асинхронными расширениями интерфейса **Windows Sockets**.

Ниже приведен фрагмент исходного кода приложения **SERVER**, демонстрирующий асинхронный прием данных [13].

После установки канала связи приложение вызывает функцию **WSAAsyncSelect**, указывая ей в качестве последнего параметрам комбинацию констант **FD_READ** и **FD_CLOSE**. При этом функция главного окна приложения будет получать сообщение **WSA_NETEVENT** в тот момент, когда чтение данных не вызовет блокировки приложения.

```
#define  WSA_NETEVENT  (WM_USER+2)

rc = WSAAsyncSelect(srv_socket,
                    hWnd,
                    WSA_NETEVENT,
                    FD_READ | FD_CLOSE);
```

При необходимости выполнения асинхронной посылки данных следует указать функции **WSAAsyncSelect** еще и параметр **FD_WRITE**.

При успешном выполнении функция **WSAAsyncSelect** возвращает **NULL**, при ошибке - **SOCKET_ERROR**.

В зависимости от значения последнего параметра могут возникать разные коды ошибки, которые можно получить с помощью функции **WSAGetLastError**. Следующие ошибки могут возникнуть при любом значении этого параметрам

Код ошибки	*Описание*
WSANOTINITIALISED	Перед использованием функции необходимо вызвать функцию **WSAStartup**
WSAENETDOWN	Сбой в сети
WSAEINVAL	Сокет не был подключен функцией **bind**
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса **Windows Sockets**

Дополнительный код ошибки можно получить, проанализировав параметр **lParam** при помощи макроса **WSAGETSELECTERROR**.

При использовании параметра **FD_CONNECT** возможно появление следующих ошибок

Код ошибки	*Описание*
WSAEADDRINUSE	Описанный адрес уже используется
WSAEADDRNOTAVAIL	Указанный адрес недоступен
WSAEAFNOSUPPORT	Для данного сокета нельзя использовать указанное семейство адресов
WSAECONNREFUSED	Попытка установления канала связи была отвергнута
WSAEDESTADDRREQ	Необходимо указать адрес получателя пакета
WSAEFAULT	Неправильно указан параметр **namelen**
WSAEINVAL	Сокет уже подключен к адресу
WSAEICONN	Сокет уже подсоединен
WSAEMFILE	Больше нет дескрипторов
WSAENETUNREACH	Из данного узла и в данное время невозможно получить доступ к сети
WSAENOBUFS	Нет места для размещения буфера
WSAENOTCONN	Сокет уже подключен

WSAENOTSOCK	Указан дескриптор файла, не сокета
WSAETIMEDOUT	При попытке установления канала связи возникла задержка во времени

При использовании параметра **FD_CLOSE** может возникнуть одна из следующих ошибок

Код ошибки	*Описание*
WSAENETDOWN	Сбой в сети
WSAECONNRESET	Разрыв соединения удаленным узлом
WSAECONNABORTED	Сбой из-за слишком большой задержки или по другой причине

В случае использования параметров **FD_READ**, **FD_WRITE**, **FD_OOB** или **FD_ACCEPT** может возникнуть ошибка с кодом **WSAENETDOWN**.

Обработчик сообщения **WSA_NETEVENT** должен выполнить анализ причины, по которой он был вызван, так как за один вызов функции **WSAAsyncSelect** можно задать несколько событий, вызывающих генерацию этого сообщения. Такой анализ проводится, например, следующим образом (параметр **wParam** содержит дескриптор сокета, на котором выполняется передача данных, а **lParam** - код происшедшего в сети события)

```
void
WndProc_OnWSANetEvent(HWND hWnd,
                      UINT msg,
                      WPARAM wParam,
                      LPARAM lParam)
{
   char szTemp[256];
   int rc;

   // если на сокете выполняется передача данных, то
   // принять и отобразить эти данные в виде строки текста
   if (WSAGETSELECTEVENT(lParam) == FD_READ)
   {
      rc = recv((SOCKET) wParam, szTemp, 256, 0);
      if (rc)
      {
         szTemp[rc] = '\0';
         MessageBox(NULL, szTemp,  "Принятые данные",  MB_OK);
      }
      return;
   }
   // если соединение завершено, вывести сообщение об этом
   else
   if (WSAGETSELECTEVENT(lParam) == FD_CLOSE)
```

```
   {
      MessageBox(NULL, "Соединение завершено",  "Server",  MB_OK);
   }
} // конец функции WndProc_OnWSANetEvent
```

В некоторых случаях целесообразно использовать протокол негарантированной доставки UDP (*User Datagram Protocol*), так как он, например, допускает одновременную рассылку пакетов сем узлам сети (режим *broadcast*). При этом не требуется создавать канал данных, поэтому процедура инициализации упрощается. Сервер и клиент UDP должны создать сокет с помощью функции socket и связать с ним адрес IP с помощью функции bind; функции connect, listen и accept не используются. Для обмена данными приложения UDP вызывают функции sendto и recvfrom (вместо send и recv соответственно).

Полный текст С-программ реальных сетевых приложений **SERVER** и **CLIENT**, фрагменты которых рассмотрены выше, приведен в [13]; любителям Pascal-программирования можно рекомендовать работу [6], также предлагающую исходные коды многих интересных сетевых приложений.

Исходные тексты практических примеров использующих сокеты сетевых приложений на языке Java приведены в работе [16].

Среды программирования Delphi/C++Builder фирмы Inprise Corp. содержат штатные компоненты, обеспечивающие функционирование употребительных протоколов, отображение информации в формате HTML и др.

Примеры исходных кодов WinSock-приложений можно найти в сети InterNet по адресам

- info.isoc.org/home.html
- www.ietf.cnri.reston.va.us/home.html
- ds.internic.net/ds/dspg/intdoc.html
- www.internic.net/std
- www.sockets.com
- www.startup.com и др.

Таким образом, интерфейс **Windows Sockets** предоставляет программисту набор функций высокого уровня (и в то же время гибких в использовании), достаточный для самостоятельного создания сетевого приложения практически любой (необходимой в конкретном случае) сложности.

7. ГЛОБАЛЬНАЯ СЕТЬ InterNet

Сеть InterNet (INTERconnection NETwork) по сути дела является синтезом многих локальных, корпоративных и национальных сетей, обладающих часто собственными внутренними (весьма разнообразными) линиями связи и

протоколами. Объединяет их в единую глобальную сеть система серверов различной аппаратной и программной конфигурации (объединяемых между собой спутниковыми и оптоволоконными линиями связи, реже коаксиальными и иными кабелями, см. рис.7.1), а также поддержка серверами протокола высокого уровня HTTP; причем совместно с ним часто используются протоколы FTP (передача файлов), MIME (передача двоичных файлов), SMTP и POP (поддержка электронной почты); транспортным протоколом между серверами и конечными пользователями сети является TCP/IP. Некоторую дополнительную информацию можно получить с сервера рабочей группы InterNet на адресе www.ietf.cnri.reston.va.us.

Для краткого названия сети InterNet часто используют термин **WEB** (*паутина*) или просто **Сеть** (с заглавной буквы), далее эти термины будут считаться синонимичными.

Подобно сети InterNet, с начала 90-х годов развиваются сети типа IntraNet. Под IntraNet понимается любая (обычно корпоративная - в масштабах предприятия) компьютерная сеть, использующая протокол TCP/IP, обычные для InterNet методы адресации и HTML-протокол передачи гипертекста, применяющая привычное для InterNet программное обеспечение (например, на стороне сервера Microsoft Information Server, а на стороне клиента стандартные WEB-броузеры); часто сеть IntraNet имеет выход в InterNet.

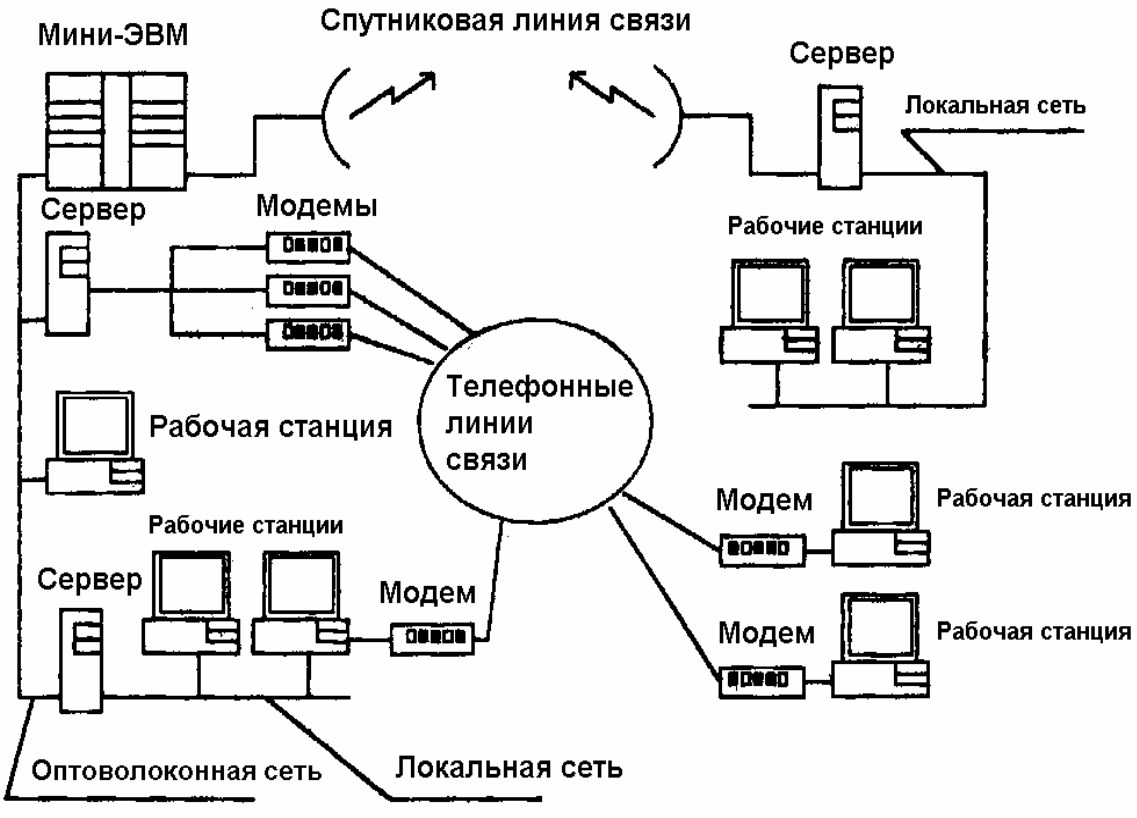

Рис.7.1. Компоненты глобальной сети.

7.1.**ИСТОРИЯ И ОСНОВНЫЕ КОНЦЕПЦИИ СЕТИ InterNet**

Проект создания глобальной сети стартовал в конце 60-х годов и финансировался правительством США в рамках военного агентства DARPA (*Defence Advanced Research Project Agency*). В результате была создана сеть ARPANET (и несколько других сетей, обслуживающих военно-космическую промышленность США).

В настоящее время сеть InterNet объединяет многие глобальные сети и насчитывает миллионы серверов.

Основные типы серверов (и предоставляемые ими услуги) приведены ниже

Тип сервера	*Предоставляемые услуги*
Сервер FTP	Хранение больших объемов файлов для передачи (выгрузки) на локальные диски пользователей (редко наоборот)
Сервер Gopher	Хранение только текстовой информации (статьи, документация, короткие заметки и т.д.) с той же целью; для поиска нужного документа используется многоуровневое меню, документы могут содержать гиперссылки
Почтовый сервер (сервер E-Mail)	Обеспечение передачи и хранения электронной почты (письма, кроме текста, могут содержать дополнительные вложения в виде произвольных файлов - звуковых, изображения и др.)
Сервер новостей (сервер News)	Хранение конференций (каждая имеет свою тему), в конференциях хранятся статьи, программные файлы, мультимедиа и др.
Сервер WWW (*World Wide Web - 'мировая паутина'*)	Хранение любой допустимой для ЭВМ информации в формате гиперссылок (допускаются ссылки как на документы внутри сервера, так и на любые документы других серверов), допускают динамически формируемую информацию и интерактивный обмен данными с удаленным пользователем

Сервером является достаточно мощная ЭВМ со специализированным *серверным программным обеспечением*, призванным эффективно обеспечивать конкретные сетевые операции.

Для операционных систем Windows'9x и Windows'NT WorkStation 4.0 часто используются серверы Personal Web Server и Microsoft Peer WebServices соответственно (для Windows'NT 4.0 Server применяется значительно более мощный сервер Microsoft Information Server), общие сведения об их установке и настройке приведены в [15]; для успешного функционирования этих серверов необходимо подключение к локальной сети по протоколу TCP/IP или к высокоскоростному каналу сети InterNet (создатели ПО успешно эксплуатируют серверы на локальной ЭВМ в целях отладки сложных WEB-сайтов; в частности, это единственный способ отладки CGI-

приложений). В среде ОС UNIX обычно используется сервер Apache (существуют версии для LINUX, Solaris, SunOS 4.x и для Windows).

Серверы сети InterNet хранят огромные объемы информации и обрабатывают запросы к этой информации для многих пользователей одновременно.

Одной из основных концепций сети InterNet является ее открытость. Это означает, что любой пользователь (затратив минимум средств) может создать свою собственную WEB-страницу, WEB-сайт (набор логически связанных WEB-страниц) или WEB-сервер, на которых может размещать произвольную информацию; причем в качестве пользовательской ЭВМ может выступать практически любой тип компьютера, оснащенный любой операционной системой.

В настоящее время сети InterNet используется для экспресс-информации, рекламы, операций купли-продажи, в банковском деле и др., в Сети размещено более миллиарда уникальных пользовательских страниц; новые применения Сети предлагаются ежемесячно.

7.1.1.**ПРИНЦИПЫ АДРЕСАЦИИ В InterNet**

Каждое подключенное к сети InterNet устройство (*узел, host*) однозначно адресуется 32-значным уникальным двоичным числом (разбиваемым точкой на 4 октета - например, 198.137.240.91). Адрес узла логически разделяется на две части, одна из которых называется идентификатором сети (Network ID), а другая - идентификатором узла (Host ID). Глобальная сеть объединяет множество сетей, каждая из которых имеет свой идентификатор Network ID, в каждой сети может находиться некоторое количество узлов, каждый из которых имеет свой Host ID. Именно таким образом (с помощью пары чисел - Network ID и Host ID) можно адресовать любой подключенный к глобальной сети на базе протокола TCP/IP узел.

Существуют несколько классов адресов (A, B, C, D...), для которых используется различная разрядность полей Network ID и Host ID.

Старшие разряды первого октета имеют специальное значение - они определяют принадлежность адреса к одному из 5 классов

Класс	*Первый октет (двоичное)*	*Второй октет (десятичное)*	*Количество сетей*	*Количество host-компьютеров на сеть*
Класс A	00000001 ÷ 01111110	1 ÷ 126	126	16 млн.
Класс B	10000000 ÷ 10111111	128 ÷ 191	16382	65534
Класс C	11000000 ÷ 11011111	192 ÷ 223	2 млн.	254

Класс D	11100000 ÷ 11110111	224 ÷ 239	группов.	группов.
Класс E	11110000 ÷ 11110111	240 ÷ 247	эксперим.	эксперим.

В адресе класса A первый разряд равен 0, следующие 7 разрядов идентифицируют сеть, а последние 24 идентифицируют главный компьютер (host-компьютер) сети. При семи разрядах в части адреса сети минус два специальных номера сети (0 и 127) в классе A может быть всего 2^2-2=126 сетей, но в каждой из них может быть до 2^{24}-2 или более 16 млн. компьютеров. Таким образом, адреса класса A используются только для большого бизнеса, в военных и исследовательских организациях (например, General Electric, Defence Intelligence Agency, AT&T Bell Laboratory, Massachusetts Institute of Technology).

Если первые два разряда адреса равны 10 - это адрес класса B, тогда последующие 16 разрядов указывают адрес сети, а последующие - адрес host-компьютера. Адреса класса B употребляются чаще адресов класса A для корпораций, университетов и поставщиков услуг InterNet.

Первые три разряда в классе C равны 110, последующий 21 разряд указывают адрес сети, последние 8 - host-компьютер. Адреса класса C употребляются организациями, у которых имеется менее 250 подключенных к InterNet устройств.

Адреса класса D, который начинаются с 110, только недавно начинают использоваться и поддерживают специальную службу групповой доставки сообщений (предназначены для компьютеров, совместно использующих общий протокол, а не для групп компьютеров, совместно использующих общую сеть). Групповая доставка сообщений InterNet находится на стадии эксперимента, но в будущем может стать основой современной широковещательной технологии, такой как радио и телевидение.

Адреса класс E начинаются разрядами 11110 и зарезервированы для будущего расширения Сети.

Некоторые адреса зарезервированы для специальных целей

- Адрес 0.0.0.0 предназначен для передачи пакетов ‘самому себе’, т.е. на свой узел.
- Адрес 127.0.0.1 используется для тестирования сетевых приложений.
- Адрес, в котором указан номер сети, а номер узла нулевой, используется для обозначения сети (например, 191.24.2.0).
- Если все биты поля номера узла равны единице (например, 193.24.2.255), то это широковещательный адрес, пользуясь которым можно передавать пакеты сразу всем узлам данной сети.

- Если все биты идентификатора сети и все биты идентификатора узла единичные (например, 255.255.255.255), адресуются все узлы данной сети.
- Для адресации узла в данной сети можно вместо номера сети указать нулевое значение (например, 0.0.0.2).

При подключении к InterNet пользователю выделяется постоянный или временный адрес (или диапазон адресов, если пользователь решит организовать собственную сеть, подключенную к InterNet). Временный адрес обычно действует лишь на время сеанса связи с Сетью посредством телефонной линии, при создании собственного сервера WWW необходим постоянный адрес (а при подключении к этому серверу через ЛВС других пользователей - некоторый диапазон адресов).

В случае затруднений при соединении с InterNet можно связаться с международной организацией InterNIC (*Internet Network International Center*) по адресу **www.internic.net** или через FTP **ftp.internic.net** или посредством электронной почты **hostmaster@internic.net.**

Дополнительная информация о функционировании подсетей приведена в документе RFC 950 ‘Internet Standart Subnetting Protocol’.

Маской подсети называют 32-разрядное число, предназначенное для выделения компонент идентификатора сети Network ID и идентификатора Node ID из полного адреса, выделение Network ID выполняется простой логической операцией ’И’ между адресом и маской подсети. По умолчанию для маски подсети используются следующие значения

Класс адреса	*Маска подсети, принятая по умолчанию*
A	255.0.0.0
B	255.255.0.0
C	255.255.255.0

Значение маски подсети указывается при настройке сетевой компоненты TCP/IP и может применяться для разделения крупных сетей на подсети (например, если имеется сеть с адресами класса B, допускающая подключение до 65534 узлов, то можно разделить ее на несколько частей путем указания соответствующих масок для подсетей). С помощью маски подсети InterNet поддерживает трехъярусную иерархию ‘сеть/подсеть/host-компьютер’ (вместо двухъярусной модели ‘сеть/главный компьютер’). Подсеть известна только на локальном уровне, остальная часть InterNet’а интерпретирует адрес попрежнему с помощью стандартного способа.

В связи с (безудержным) расширением сети IntrNet в последнее время ощущается нехватка IP-адресов; проблему призван решить стандарт IP версии 6 (сокращенно IPv6). Протокол IPv6 подвергается критике за включение во все IP-адреса уникального кода, однозначно идентифицирующего вклю-

ченный в Сеть компьютер (таким образом может быть идентифицирован конкретный пользователь; ср. с системами ЭШЕЛОН и СОРМ, подраздел 3.2).

Однако IP-адреса, задаваемые в виде 4-х десятичных чисел, неудобны для восприятия человеком. Поэтому была разработана т.н. *доменная система имен узлов*, обеспечивающая уникальность имен за счет иерархической структуры (рис.7.2); при этом полный доменный адрес формируется *справа налево* путем добавления имен вложенных доменов, разделенных точкой. Регистрацию доменного имени осуществляет уже упоминавшаяся организация InterNIC, регистрация платная. Дополнительная информация - RFC 1034 ‘Domain Names - Concepts and Facilities’ и RFC 1035 ‘Domain Names - Implementation and Specification’.

Для отображения доменных имен на адреса IP в сети InterNet существует специальная распределенная база данных DNS (*Domain Name System*), пользуясь которой (через т.н. сервер DNS) узлы могут преобразовывать доменные адреса в численные IP-адреса. Для ОС Windows применяются также серверы WINS, управляющие базой данных, в которой устанавливается соответствие адресов TCP/IP и имен компьютеров NetBIOS сети Microsoft. Для установления соответствия между адресами IP и доменными адресами используется файл HOSTS (соответствия для NetBIOS-адресов задаются файлом LMHOST); оба файла редактируются вручную.

При создании сети InterNet для нее было определено несколько доменов высшего уровня, разделявших доменные адреса по их принадлежности к различным организациям (применяются в основном организациями США)

Имя домена	*Организация*
gov	Правительственные организации
mil	Военные организации
com	Коммерческие организации
org	Некоммерческие организации
edu	Исследовательские организации, учебные заведения
net	Занимающиеся сетевыми технологиями организации

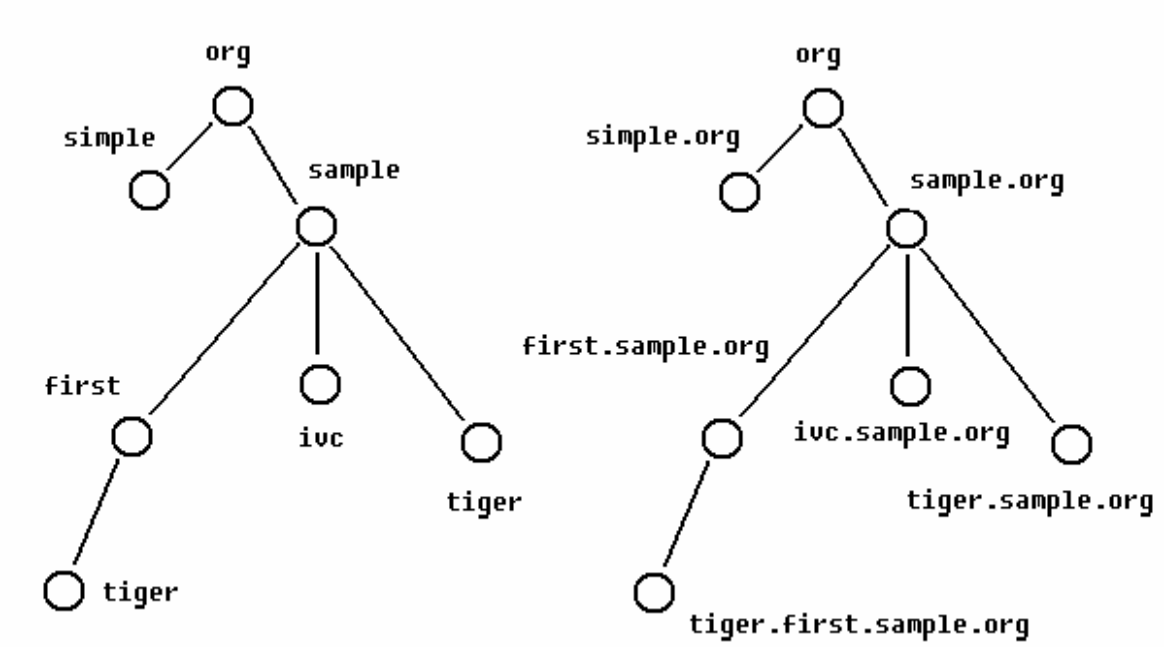

Рис.7.2. Пример иерархической доменной структуры системы имен (слева) и полные доменные имена узлов (справа).

По мере дальнейшего развития сети InterNet в ней появились домены верхнего уровня, принадлежащие различным странам (полный список находится на сервере **ftp.wisc.edu**)

Имя домена	*Страна*
au	Австралия
de	Германия
ru	Российская Федерация
ua	Украина
us	США

В России до сих пор используется имя домена **su**, принадлежащего в прошлом СССР.

Адрес каждого ресурса (файла) InterNet задается с использованием т.н. адреса URL (*Uniform Resource Locator*), имеющего следующий формат:

протокол://доменный_адрес_узла/путь_к_файлу/имя_файла

Как видно из вышеприведенного описания, URL-адрес по синтаксису близок к принятым в современных ОС полным (с учетом пути по файловой системе) адресам файлов и является расширением этой системы, дополнительно введена информация о протоколе обмена сообщениями и понятие узла сети. URL-адрес однозначно определяет конкретный файл в Сети, причем для пользователя абсолютно неважно, находится ли этот файл на данной ЭВМ

или на компьютере, расположенном на расстоянии многих тысяч километров и включенном в сеть InterNet.

Для серверов WWW применяется следующая форма универсального идентификатора ресурсов URL

http://host[:port][path]

Параметр *host* обязателен. Он должен быть указан в виде доменного адреса или как IP-адрес (в форме четырех десятизначных чисел), например

http://www.microsoft.com
http://154.23.12.101

Необязательный параметр *port* задает номер порта для работы с протоколом HTTP, по умолчанию это порт с номером 80. Номер порта идентифицирует программу, работающую в узле сети TCP/IP и взаимодействующую с другими программами, функционирующими на том же или на другом узле сети, подробнее см. [13]. Ниже показано, как нужно указывать в URL-адресе номер порта

http://www.my_server.srv/:82

Для адресов электронной почты используется несколько иной (использующий символ '@' между именем почтового ящика и доменным именем узла) синтаксис адреса (подробнее см. следующий подраздел).

7.2.СТАНДАРТНЫЕ ПРИЛОЖЕНИЯ ДЛЯ РАБОТЫ С InterNet

Ранее (см. рис.7.1) были приведены основные компоненты сети InterNet, ниже будут рассмотрены программные приложения, позволяющие работать с Сетью конечным пользователям.

В большинстве случаев пользователь подключается к Сети через телефонную линию с помощью специального устройства - *модема* (*сокращение терминов 'модулятор-демодулятор'*), при этом скорость обмена данными определяется типом модема и (в большей степени) качеством телефонного канала и обычно составляет 14400÷57600 бит в секунду (бод), сервер же 'общается' с Сетью через высокоскоростной канал (на рис.7.3 и 7.4 в качестве примера показан спутниковый, причем скорость обмена в этой части канала на порядки выше). Т.о. именно скорость передачи данных к конкретному компьютеру обычно лимитирует пропускную способность сети (т.н. 'проблема последней мили').

Повышенную скорость сетевого обмена обеспечива*ют выделенные линии связи* (часто те же жилы телефонного кабеля, специально выделенные для передачи информации по Сети), при этом удается повысить скорость обмена не менее чем на порядок; технология DSL (*Digital Subscrabe Line*) подразумевает использование модулируемых ультразвуковых частот в обычном телефонном канале, обеспечивает скорость обмена до 7,5 Мбит/сек и не мешает обычным телефонным переговорам.

Часто соединенные локальной сетью компьютеры одного учреждения подключаются к сети InterNet через выделенную ЭВМ со специализированным ПО, называемую в этом случае *proxy-сервером* (рис.7.4). Proxy-сервер (‘доверенный компьютер’) часто обеспечивает выполнение некоторых вспомогательных функций (например, ограничение на д**о**ступ к Сети определенных пользователей ЛВС, функции фильтрации сообщений и др. - т.е. выполняет функции *брандмауэра*).

В состав ОС Windows включены компоненты, поддерживающие связь по протоколу TCP/IP с использованием телефонного (или другого) канала связи, устанавливаемые опционально (по желанию); настройка этого ПО подробно описана, например, в работе [13].

Одним из наиболее простых распространенных сетевых приложений сети InterNet являются программы обеспечения *электронной почты* (E-Mail). Для того, чтобы послать электронное письмо, необходимо иметь электронный адрес своего корреспондента; узнать его можно при личной встрече, по телефону, из рекламы (в том числе на WEB-страницах). Таким образом, любой пользователь имеет возможность послать (электронное) письмо по известному адресу, но только владелец почтового ящика имеет возможность просматривать и удалять письма (используется парольная система).

Электронные адреса E-Mail выглядят следующим образом

Адрес E-Mail	*Назначение*
vep@oktava.msk.su	Вычислительный центр МГАПИ
e881@yahoo.com	Почтовый ящик автора методического руководства
info@daidocomp.com	Фирма DaidoSeiko
frolov@glas.apc.org	Личный почтовый ящик

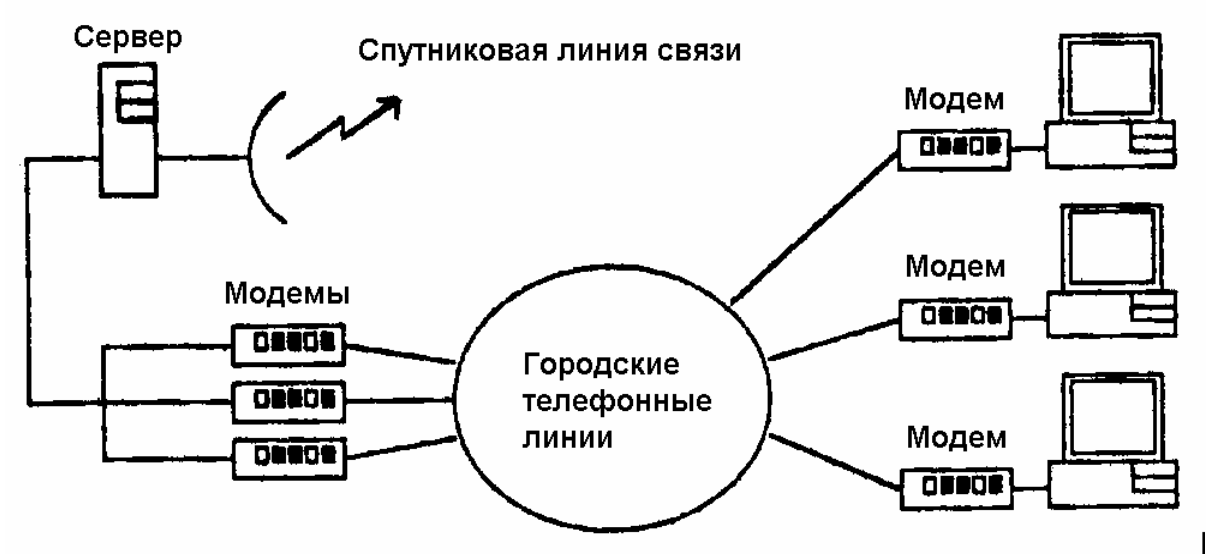

Рис.7.3. Подключение к серверу через городские телефонные сети.

При отправке письма по E-Mail адрес отправителя обычно добавляется автоматически. Большинство электронных писем представляет собой текстовые сообщения объемом не более нескольких килобайт, но современные программы допускают включение и двоичных файлов (изображения, программы, документы WinWord и др.). На рис.7.5 приведена копия экрана дисплея ПЭВМ при работе почтовой программы Microsoft News and Mail.

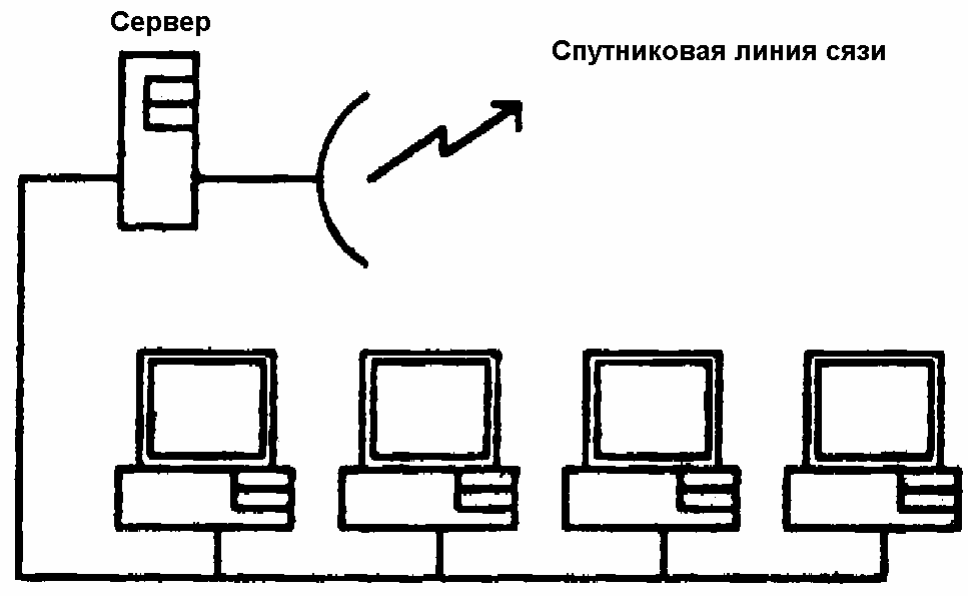

Рис.7.4. Подключение к InterNet через локальную сеть.

Для посылки письма следует ввести с клавиатуры адрес получателя в графе ‘Кому’ (‘To’ в английской транскрипции), тему письма в графе ‘Тема’ (‘Subject’) и набрать текст послания; дополнительно в графе ‘Копия’ (‘Cc’)

можно указать адреса по**о**сылки копий письма и прикрепить к посланию двоичные файлы (для данной программы путем нажатия кнопки с символом ‘скрепка’ с последующим выбором нужного файла).

Прием писем реально происходит при подсоединении программы электронной почты к почтовому серверу, список полученных писем изображается в таблице, владелец почтового ящика имеет возможность просмотреть письма, ответить на любое письмо или удалить его.

Среди других популярных почтовых программ можно отметить систему Eudora for Microsoft Windows, Microsoft Exchange и др. [13].

Значительный интерес представляют т.н. *электронные конференции* (телеконференции), или *сетевые новости*. В Сети имеются серверы электронных конференций, которые хранят статьи (в виде текстовых документов), объединенные в группы. Имеющие доступ к такому серверу пользователи могут посылать в выбранные ими группы свои статьи и просматривать и получать статьи, записанные туда другими пользователями. Группы существуют по интересам (например, пользователи Borland Delphi, Interbase, Oracle и др.), постоянно появляются новые. Пользователи могут обсуждать проблемы конференции, задавать вопросы и оказывать помощь нуждающимся; большинство конференций транслируется по всей сети InterNet (существуют и локальные конференции); для работы с электронными конференциями пользователь обычно должен *подписаться* на интересующую его конференцию (или группу новостей). Одним из удобных приложений для работы с электронными конференциями является News Express, позволяющее легко выбирать нужную конференцию, просматривать список статей (с обеспечением сортировки по теме, дате, имени автора или размеру статьи).

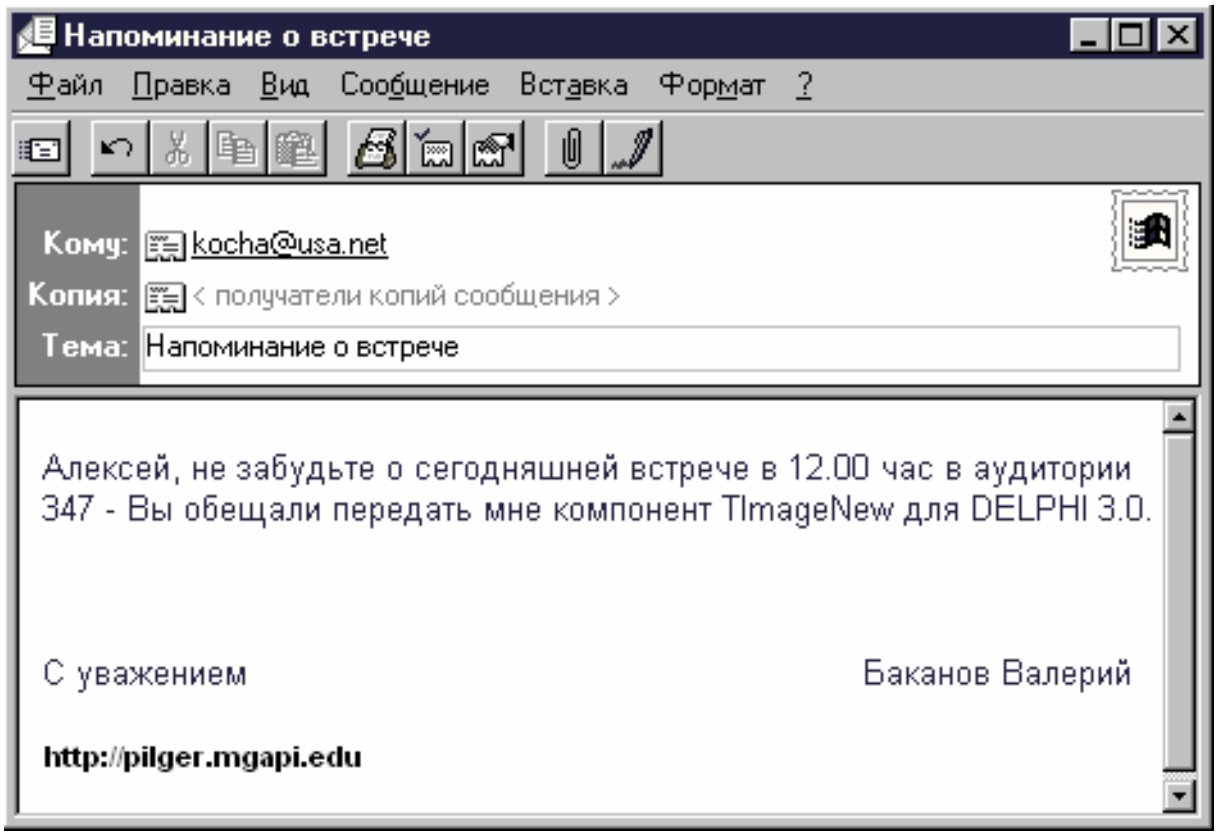

Рис.7.5. Вид экрана ПЭВМ при работе с почтовой программой Microsoft News and Mail.

С помощью программы Telnet (*Terminal Access to a remote host*) имеется возможность подключения к консоли удаленной ЭВМ (что удобно, например, администратору сети и позволяет управлять работой сервера, не выходя из дома). Сетевое приложение HyperTerminal (фирма *Hilgraeve Inc*.) позволяет обмениваться файлами с удаленной ЭВМ по телефонной сети.

Возможность ‘разговаривать’ через InterNet путем набора и считывания с экрана текстовых посланий является уже привычным (и скучным) сервисом (для ‘оживления’ беседы программный продукт Microsoft Chat вводит в действие асоциируемых с конкретными пользователями типизированных персонажей, ‘разговаривающих’ на фоне определенного пейзажа). Популярная система ICQ (звуковое подражание фразе ‘I Seek You’ - ‘Я ищу тебя’, компания Mirabilis, Тель-Авив, 1996, бесплатная до 2000 г.) позволяет обмениваться персональными сообщениями с конкретным человеком в режиме реального времени (‘двухсторонний пейджер’, подробнее см. www.icq.com и www.dir.ru/internet/icq/russian/index.htm).

Существует ПО для обеспечения реального разговора через InterNet (т.н. IP-телефония), причем такой разговор обходится существенно дешевле, чем при использовании международного телефона; все чаще Сеть используется для передачи реальных видеоизображений. Большие размеры аувидеофайлов и ограничения полосы частот Сети для передачи реального аудио и

видео вызвали появление технологии потоковой (*streaming*) передачи данных, для передачи голосовых сообщений в последнее время предлагается вышеупомянутая в подразделе 3.4 технология MPLS, позволяющая информационным пакетам ‘обходить’ перегруженные узлы Сети (тем самым минимизируя недопустимые при передаче голосовых сообщений задержки).

Получение файлов из сети InterNet обычно производится с помощью протокола FTP и специализированных приложений, например, FTP-32 Client for Windows (рис.7.6), соответствующие возможности включены во многие программные оболочки (например, Norton File Manager и Windows Commander).

Адрес целевого FTP-сервера (наряду с некоторыми дополнительными параметрами) вводится в окне, появляющемся сразу после загрузки приложения FTP-32 Client for Windows, при работе с программой пользователь на левой панели видит диски и каталоги собственного компьютера, на правой - удаленного; существуют возможности перемещения по каталогам файловой системы, выделения нужного файла и пересылки выбранного файла (кнопки со стрелками), причем пос**ы**лка пользовательского файла на FTP-сервер разрешена лишь привилегированному пользователю (с целью устранения переполнения серверного набора дисковых томов ненужными файлами и предотвращения распространения компьютерных вирусов)**.**

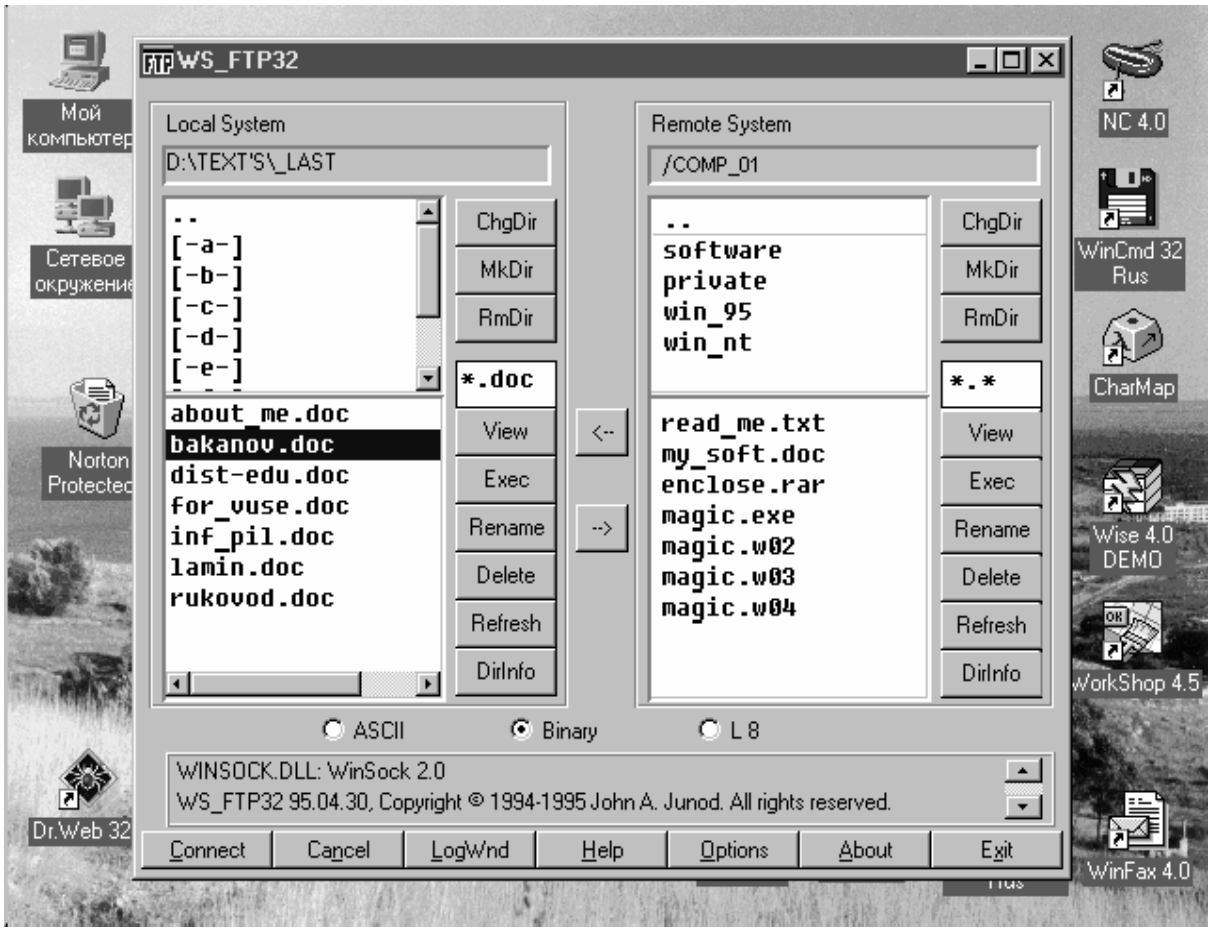

Рис.7.6. Вид экрана ПЭВМ при работе с приложением FTP-32 Client for Windows.

Наиболее мощные программные средства сети InterNet - программы обеспечения работы с серверами Word Wide Web (WWW), эти приложения часто называют *броузерами* (*browser*) или *навигаторами* (*navigator*).

Серверы WWW хранят информацию в виде *гипертекстовых файлов* (см. следующий подраздел), броузеры по сети получают (с использованием протокола HTTP) такой файл (соответствующий *странице гипертекста* или *WEB-странице*), специальным образом интерпретируют его и, при необходимости, отсылают введенную пользователем информацию назад серверу (например, запрос к базе данных); общая схема взаимодействия броузера с сервером WWW приведена на рис.7.7.

Для просмотра страниц WEB-страниц первоначально были созданы броузеры Mosaic (разработка NCSA, *National Center for Supercomputing Applications*, Illinois University, 1993), Cello (LII, *Legal Information Institute*, Cornell Law School), Lynx и др., в настоящее время наиболее часто используются Microsoft Internet Explorer, Netscape Internet Navigator, Opera (Opera Software, www.opera.com) для эксплуатации в среде различных операционных систем; ниже будут рассмотрены наиболее распространенные броузеры фирм Netscape Communications Corp. и Microsoft Corp.

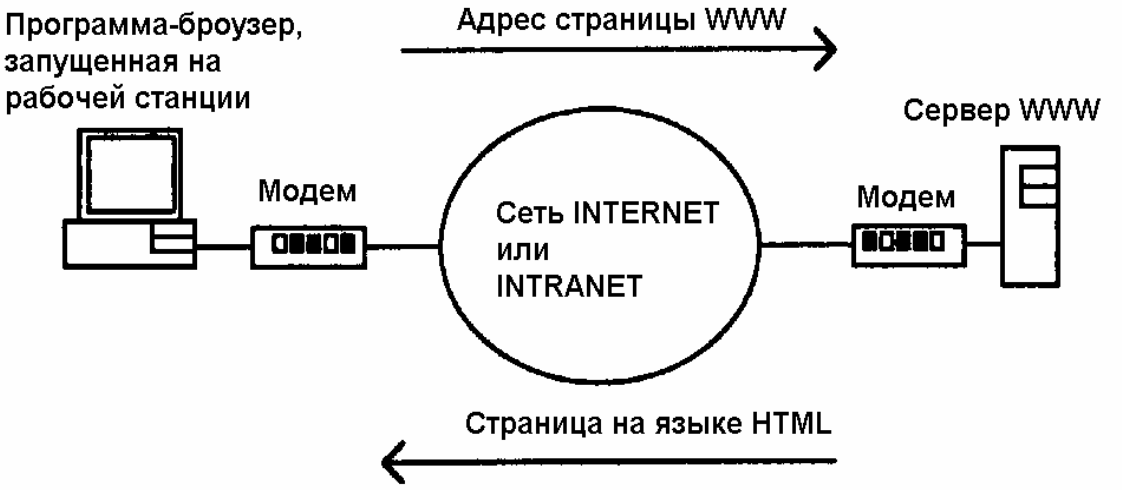

Рис.7.7. Взаимодействие броузера и сервера WWW.

На рис.7.8 приведена копия экрана ПЭВМ во время функционирования приложения Netscape Communicator, на рис.7.9 - приложения Microsoft Internet Explorer (в обеих системах целевой WWW-адрес вводится в строке ввода, расположенной в верхней части окна броузера). Броузер фирмы Netscape Communications Corp. появился исторически первым, однако Microsoft Corp. включила свой броузер в состав ОС собственной разработки (что уже ряд лет служит причиной судебного разбирательства между указанными фирмами). Одними из последних нововведений фирмы Microsoft Corp. являются динамический HTML (DHTML, не поддерживаемый броузерами сторонних фирм) и новая компонентная модель разработки для WEB, под названием скриплеты (*scriplets*, понятие составлено из слов ‘сценарии’ и ‘аплеты’).

Оба приложения снабжены развит**о**й системой навигации и кеширования WEB-страниц, поддерживают протоколы TCP/IP, HTTP, FTP, обеспечивают E-Mail и работу с FTP-сервер**а**ми, содержат (встроенные) средства шифрации, однако только в последних версиях Microsoft Internet Explorer’а появилось средство *публикации* (перес**ы**лки с ЭВМ разработчика на WWW-сервер) файлов описания WEB-страниц, также первой фирма Netscape Communications Corp. выпустила средство для создания и редактирования HTML-страниц. Версии броузеров упомянутых фирм можно получить по адресам home.netscape.com и www.microsoft.com/ie соответственно. Представляющими интерес проектами являются Mozilla (термин является видоизмененным в сторону модной ‘динозавровской’ тематики названием легендарного броузера Mozaic, см. mozilla.ru) и BackOffice (backoffice.ru).

При проверке корректности настройки сети и протокола TCP/IP полезны программа PING (тестирование связи с любым InterNet-узлом), PM Ping (то же самое для операционной системы OS/2 Warp), NETSTAT (получение информации о установленных соединениях), ROUTE (работа с таблицей мар-

шрутизации), FTP (утилита с командной строкой для загрузки файлов из Сети), FTP-PM (то же самое для OS/2 Warp), подробнее о эксплуатации этих приложений см. работу [13].

Виртуальная реальность в InterNet пока является новинкой (вследствие ограничений на объем передаваемой через Сеть информации предлагается хранить библиотеки описаний объектов виртуальной реальности на локальном компьютере, а из Сети получать лишь сценарии действий в виде текстовых файлов). При этом используется (интерпретируемый) предложенный в 1995 г. язык VRML (*Virtual Reality Modeling Language)* - язык моделирования виртуальной реальности; в настоящее время действует стандарт VRML97, практически идентичный языку VRML 2.0; на 2002 г. запланирована следующая версия VRML, получившая название X3D [22]. VRML предлагает способ создания трехмерных виртуальных миров (‘аватаров’) с возможностью ‘путешествий’ по ним в WEB; объекты VRML включают в себя текст, изображения, аудио, видео и Java-приложения (примеры VRML см. на InterNet-адресах www.webmaster.com/vrml, webspace.sgi.com, vrml.wired.com). Для общения InterNet-пользователей предлагаются поддерживающие VRML продукты Worlds Chat (фирма Worlds, Inc., www.worlds.net), Prospero’s Global Chat (www.prospero.com), Internet Round Table Society’s WebChat (www.irsociety.com) и др.

Для просмотра VRML-мир**о**в в настоящее время предлагаются несколько (бесплатных) расширений стандартных WEB-броузеров - модуль Live3D и CosmoPlayer (компания *Cosmo Software*, www.cosmosoftware.com) для броузера Nestcape Communicator, WorldView 2.0 (platunim.com) для MS Internet Explorer 5.0; удобным является VRML-клиент Cortona (фирма *ParallelGraphics*, www.paragraph.ru).

Так как описание VRML представляется обычным текстовым файлом (несложные примеры см. на адресе www.srcc.msu.su/vrml), создавать (и редактировать) исходные VRML-тексты можно с помощью простейших текстовых редакторов класса NotePad или WordPad; из специализированных редакторов VRML-кодов можно указать, например, VRMLPad фирмы *ParallelGraphics* (в**ы**грузка с узла www.paragraph.ru). Многие программы трехмерного моделирования (например, всем известный пакет 3D Studio Max) позволяют работать с данными в форме VRML.

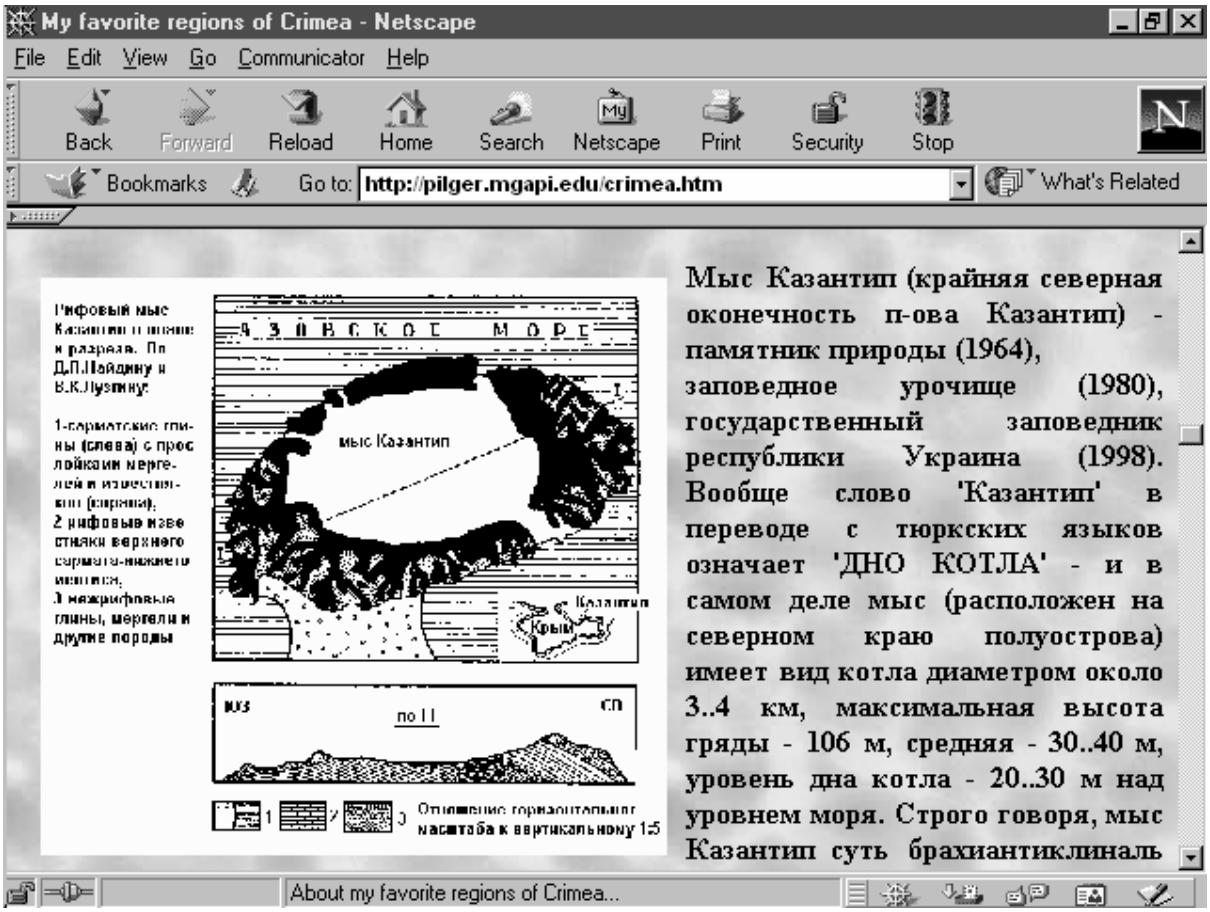

Рис.7.8. Вид экрана ПЭВМ при работе с приложением Netscape Communicator версии 4.5.

7.2.1. ЯЗЫК ОПИСАНИЯ СЦЕНАРИЕВ HTML И ЕГО РАСШИРЕНИЯ

Как было сказано выше, броузер запрашивает с WWW-сервера определенную WEB-страницу (фактически определенный HTML-файл), сервер же посылает данную страницу, а броузер (кроме поддержания связи с сервером) интерпретирует HTML-код и отображает страницу на экране.

HTML (*Hyper Text Markup Language*) - это язык *тегов* (заключенных в угловые скобки кратких предписаний, определяющих параметры отображения информации). Созданная с помощью HTML страница отображается на различных экранах (VGA, SVGA и др.) не абсолютно одинаково, однако с соблюдением некоторых общих правил отображения. Язык HTML произошел от ранее разработанного мощного языка разметки текста SGML (*Standard Generalized Markup Language*). Развитием языка HTML является XML (*eXtended Markup Language*); для обмена XML-документами между WEB-узлами разработан протокол ICE (*Information and Content Exchange*).

Метод передачи документов с помощью HTML не дает возможности полного согласования вида документа на различных платформах - гарантиру-

ется только его структура. Именно предельная простота HTML (заключающаяся в его текстовом формате и использовании коротких легкозапоминающихся инструкций) позволила HTML стать основным языком описания WEB-страниц для самых различных платформ (от Windows до UNIX, Solaris, Mac OS и др.).

HTML - очень простой язык для разметки страниц, использующий исключительно текстовые команды (при этом для выполнения некоторых команд, например, загрузки файлов изображений, мультимедиа и др. используется двоичный формат данных). В настоящее время для создания HTML-файлов (т.е. разработки WEB-страниц) применяется специальное ПО - Netscape Composer, Microsoft FrontPage, Internet Assistant for Microsoft Office, Microsoft Word Internet Assistant, Microsoft Excel Internet Assistant, Microsoft PowerPoint 95 Internet Assistant и др. Несмотря на отсутствие строгого соблюдения правила WYSIWYG (*What You See Is What Your Get* - Что видишь, то и получишь) в цикле создания/использования WEB-страниц вышеупомянутые системы практически применяют его.

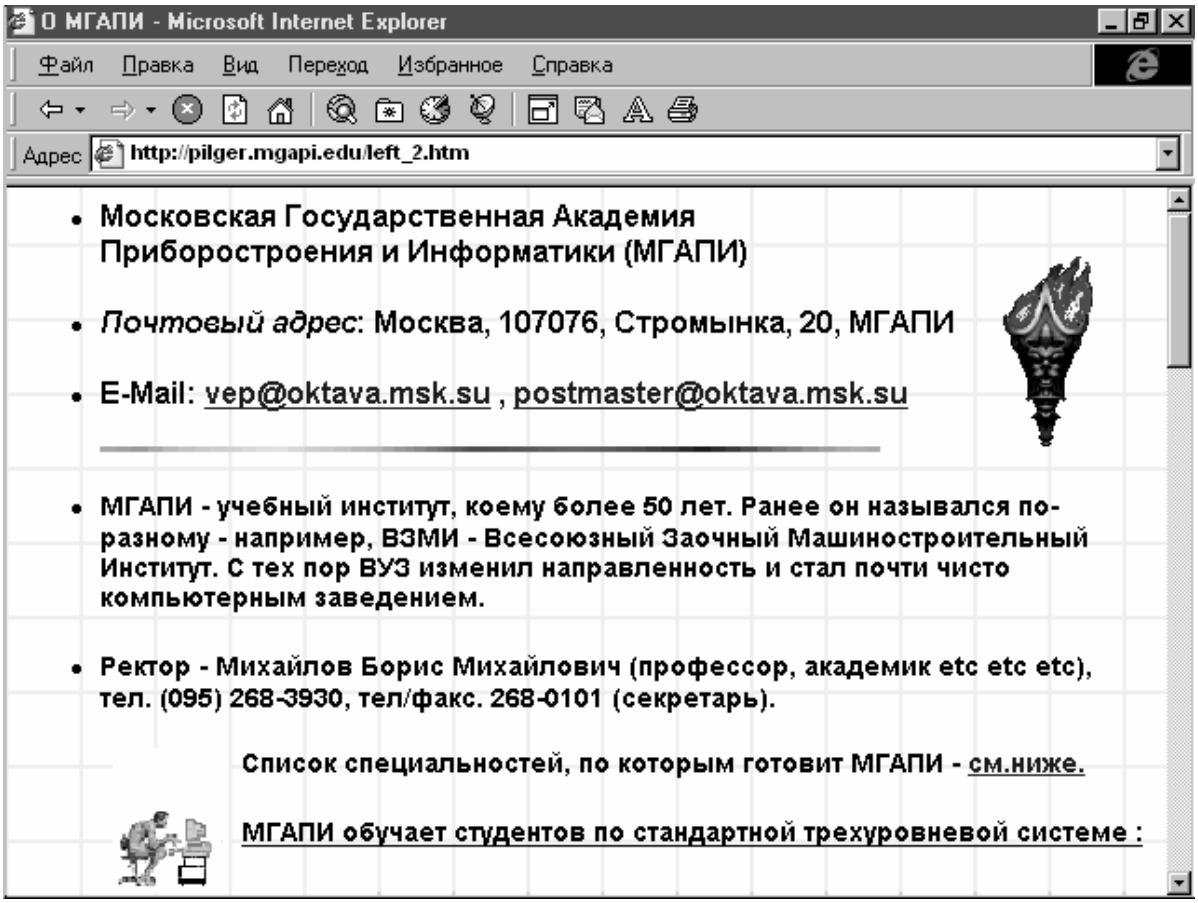

Рис.7.9. Вид экрана ПЭВМ при работе с приложением Microsoft Internet Explorer версии 4.0.

В настоящее время действуют версии 3.0 и 3.2 языка HTML, поддерживаемые броузерами обеих ведущих фирм.

Простейший вариант HTML-файла приведен ниже:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>My first WEB-page</TITLE>
</HEAD>
<BODY>
Добро пожаловать на мою первую WEB-страницу !
</BODY>
</HTML>
```

Как видно из приведенного текста, любой HTML-файл начинается тегом <HTML> и кончается </HTML>и включает блок <BODY>...</BODY> (символ наклонной черты перед именем тега ограничивает его область действия), теги могут иметь атрибуты.

После сохранения этой информации в файл FIRST.HTM (используется также расширение имени файла HTML), запуска броузера Microsoft Internet Explorer и выполнения вариантов ‘Файл|Открыть’ из главного меню броузер интерпретирует вышеприведенный HTML-код в приведенном на рис.7.10 виде.

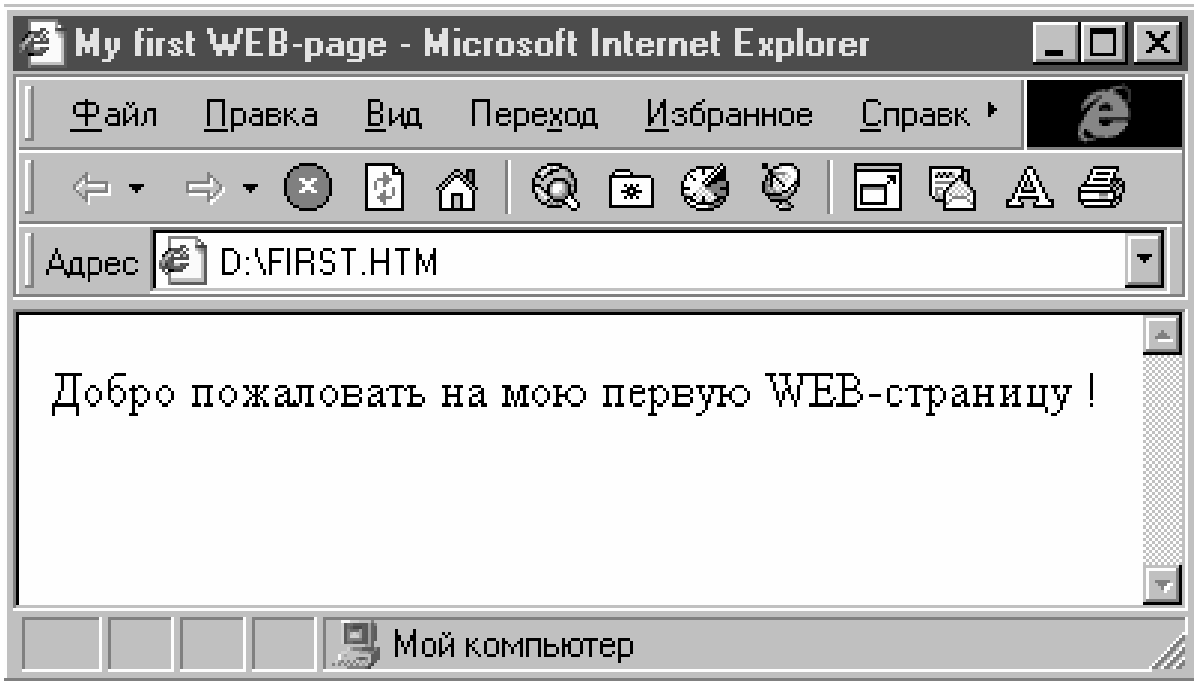

Рис.7.10.Результат отображение броузером простейшей HTML-страницы.

Важным свойством языка HTML является возможность описания *гипер-переходов* между страницами (и/или отдельными точками на одной и той же странице), именно это определяет огромные возможности HTML по представлению информации со сложными логическими связями.

Ниже приведен пример строки с возможностью перехода по гиперсвязи (при интерпретации броузером выделенная тегами <A...> и </A> часть строки отображается в отличном от других частей строки цвете, при наведении курсора ‘мыши’ на эту выделенную цветом часть строки и щелчке левой кнопкой ‘мыши’ броузер перейдет к странице (фактически загрузит новый файл) с именем ABOUT_ME.HTM (имя указано в параметрах тега <A>):

```
Добро пожаловать на <a href="about_me.htm">мою</a>
вторую WEB-страницу !
```

Другой пример - переход к метке (‘якорю’) MY_BORN на странице ABOUT_ME.HTM

```
Добро пожаловать на <a href="about_me.htm#my_born">мою</a>
третью WEB-страницу !
```

В HTML определены теги гиперпереходов по части текста, изображения, элементам мультимедиа, существует большой набор тегов описания стилей текста, расположения и масштабирования изображений и др. Ниже приведен пример использования URL-адресации в HTML-тексте

```
<img src="http://pilger.mgapi.edu/gif_89/crim_ani.gif">
```

В данном случае броузер отобразит файл CRIM_ANI.GIF, находящийся в подкаталоге GIF_89 WEB-сайта pilger.mgapi.edu (естественно, если этот сайт доступен по Сети); частным случаем является ссылка на файл, расположенный на локальной машине (так часто и разрабатываются WEB-сайты).

Ссылки на WEB-сервер (или сайт) и на адрес электронной почты сответственно оформляются в виде

```
<a href="http://pilger.mgapi.edu ">Ссылка на WEB-сайт</a>

<a href="mailto:vep@oktava.msk.su">Ссылка на E-Mail</a>
```

Возможность разрешения глобальных гиперссылок еще больше расширяет мощь InterNet, фактически появляется возможность создания *распределенной системы* хранения (и обработки) данных.

Средством обеспечения диалога в Сети служат *формы* - специализированные конструкции языка HTML, позволяющие пользователю вводить данные в отображаемые броузером поля и пересылать их серверу; для желающих серьезно ознакомиться с возможностями HTML можно рекомендовать работы [6,13,15,19].

Фирма Microsoft Corp. объявила поддержку своими броузерами тега *include* в формате гипертекстовых файлов (требуемое расширение имени файла - STM).

Однако у языка HTML есть и недостатки (вытекающие, впрочем, из его первоначального предназначения - HTML разрабатывался как язык разметки страниц и не более) - например, средствами HTML невозможно производить даже простейшие арифметические вычисления. Часто применяются следующие расширения языка HTML (выполняются на стороне клиента и/или сервера и расширяют их возможности)

- **Java** - машинно-независимый язык программирования (подробнее см. подраздел 7.2.2).
- **JavaScript, VBScript и PerlScript** - языки программирования, интегрированные в HTML-код (см. подраздел 7.2.3).
- **CGI** и **ISAPI** - серверные расширения HTML, служащие для организации полномасштабного диалога в Сети (см. подраздел 7.2.4).
- **ActiveX** - предложенная Microsoft Corp. технология, позволяющая вводить в WEB-страницы любые (выгружаемые из Сети) активные программные объекты; реализована в виде построенного на Win32 и OLE API (подробности можно получить с сервера фирмы-разработчика www.microsoft.com). Технология ActiveX, например, удачно применена фирмой Inprise Corp. для создания ‘тонкого’ клиента при работе с базами данных (внешне имеет много общего с технологией MIDAS той же фирмы, однако выполняется в броузере клиента) для InterNet’а [7].
- **IDC** - *Internet Database Connector* - средство доступа к базам данных через ODBC (*Open Database Connectivity*) в технологии ‘клиент/сервер’ для Microsoft SQL Server (IDC включает также соответствующий формат файлов).
- **ASP** (Active Server Pages) - технология позволяет решать те же задачи, что и с помощью CGI и ISAPI, однако при этом заметно упрощается процесс разработки WEB-приложений [23]. Документ ASP включает шаблон, использует серверные сценарии на языке JScript или VBScript, запросы к БД на SQL и COM-технологию.
- **PHP/FI** - развивающийся в последнее время язык создания домашних WEB-страниц, облегчения разработки форм и таблиц и анализа запросов SQL; предложения PHP/FI (одна из распространенных версий - 2.0) встраиваются непосредственно в текст HTML-страниц и выполняются серверным процессом (см. подраздел 7.2.4).

Следует отметить существующие технологии работы с мультимедиа в InterNet (работа [19] и др.)

- **RealAudio** (фирма Progressive Network) - один из примеров многообещающей технологии потоковой передачи данных (*streaming audio*), позволяющей проигрывать (звуковой) файл в процессе его загрузки (информация на адресе www.realaudio.com); одна из популярных программ - SOX (*SOund eXchange*, см. www.spies.com/Sox/). Другие примеры - системы IWave (сокращение от *InternetWave*, фирма VocalTec, информация на адресе www.dspg.com) и TrueSpeech.
- **MBONE** - *Multicast Backbone* - виртуальная сеть, позволяющая передавать видеоизображения и аудио через InterNet с использованием технологии групповой пересылки (*multicasting*). Видео воспроизводится со скоростью 1 кадр/сек (при пропускной способности канала 128 Кбайт/сек), для высококачественной передачи голоса требуется 32 или 64 Кб/сек (MBONE применялась для репортажей непосредственно с места событий, например, при выходе космонавтов в открытый космос). Работу с MBONE поддерживают приложения NetVideo, VisualAudioTool и Whiteboard (все для UNIX).
- **StreamWorks** - разработанная фирмой Xing Technology Corp. система передачи видео и аудио по любой сети (информация на www.xingtech.com). Корпорации NBS и Reuters применяют StreamWorks для передачи коммерческим подписчикам репортажей с места событий, несколько WEB-радиостанций используют StremWorks для высококачественной трансляции музыки (от 14,4 Кб/сек до 44,1 Кб/сек для стерео и 112 Кб/сек для полноценного видео).
- **VRML** - *Virtual Reality Modeling Language* - язык моделирования виртуальной реальности, подробнее см. работу [22].

В конце 90-х годов разработан протокол (именуемый WAP - *Wireless Application Protocol*) передачи информации из Сети на мобильный аппарат (в конце 1999 года Microsoft Corp. выпустила поддерживающий WAP броузер Mobile Explorer). Некоторые InterNet-компании заявили о поддержке этого (несовместимого с существующими) стандарта (например, сайт **wap.infoart.ru** фирмы Infoart Stars, ‘увидеть’ который в начале внедрения технологии можно было лишь с помощью специализированного сотового телефона. Проблему несовместимости решила компании Motorola (создавшая первый основанный на Java телефон, обеспечивающей полноценный д**о**ступ к стандартным HTML-сайтам в Сети) и другие. Почтовые услуги для владельцев телефонов с протоколом WAP предоставляет также сервер www.iname.ru и др.

Для практического использования полезным является страница www.primorye.ru/noc/ping.asp, дающая возможность проследить прохождение пакетов к выбранному узлу (ан**а**лог утилиты *ping* в глобальной Сети);

еще более показательным является визуальный трассировщик VisualRoute (www.visualware.com/download/index.html).

7.2.2. **ЯЗЫК Java ПРОГРАММИРОВАНИЯ В СЕТИ InterNet**

Связанные с наличием большого количества (трудно совместимых между собой) аппаратных платформ и операционных систем трудности при переносе разработанного ПО между различными ЭВМ инициировали разработку машинно-независимых языков программирования, одним (достаточно удачным) примером такого языка является предложенный в 1995 году фирмой Sun Microsystems язык Java (создан на основе языка Oak).

Язык Java по синтаксису бл**и**зок к C++, однако имеет существенные особенности

- Не поддерживается перегр**у**зка операторов (вследствие трудности поддержки и относительно редкого использования на практике).
- Запрещено множественное наследование (основания подобны вышеприведенным, запрет несколько смягчается возможностью использования унаследованных интерфейсов).
- Исключены указатели, являющиеся частой причиной труднолокализируемых ошибок в C++.

Метод достижения машинной независимости (переносимости) заключается в трансляции исходного Java-текста в *байт-код* (поток команд воображаемого процессора, известного как JVM - *Java Virtual Machine*), перес**ы**лке байт-кода по Сети и последующей его интерпретации (существуют и компиляторы) на конкретной ЭВМ (см. рис.7.11). В настоящее время Java перенос**и**м между компьютерами с операционными ср**е**дами Solaris, Windows'9x, Windows'NT, OS/2 и ОС для ПЭВМ Apple Macintosh, поддерживается броузерами фирм Netscape и Microsoft.

Платформа Windows

Платформа Apple Macintosh

Изменение исходного текста программы

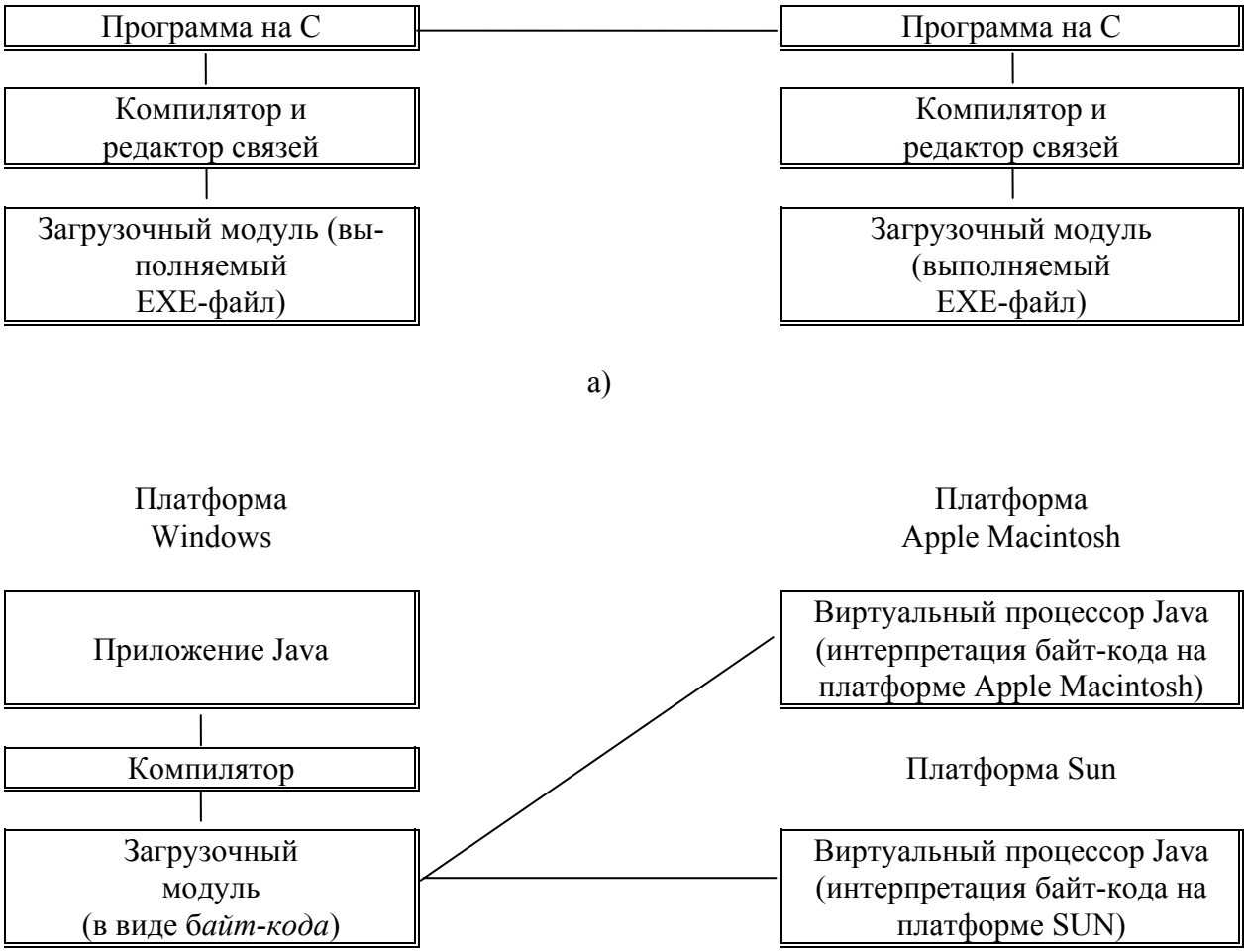

Рис.7.11.Этапы создания и переноса приложений, разработанных с помощью традиционных языков программирования (а) и приложений Java (б).

Java одновременно является и языком и набором библиотек классов, специально разработанных для применения в InterNet. Мощь языка Java основана на сочетании четырех его характеристик

1. Сетевой осведомленности (*network awareness*) - заключается в том, что каждое решение при разработке языка принималось таким образом, чтобы предоставить возможность Java-приложениям обладать сетевыми характеристиками (в применении к InterNet это вылилось в обеспечении гнездового интерфейса низкого уровня - *low-level socket interface*).
2. Переносимости - возможность исполнения Java-программ на различных аппаратных и операционных платформах.
3. Безопасности - достигается путем сегментации памяти, выгрузки из Сети только классов Java (а не ‘родных’, зависящих от конкретной ОС, методов) и фильтрования всех потенциально опасных требований д**о**ступа к системным ресурсам через специальный класс SecurityManager.

4. Объектной ориентации - путем разграничения д**о**ступа к объектам путем применения концепций *наследования* и *инкапсуляции*.

Java-программа может быть *автономным приложением* или *апплетом* (маленькой программой, выполняющейся внутри другой, более крупной программы - броузера; в отличие от автономных приложений апплеты не имеют д**о**ступа к файловой системе локального компьютера), ниже приведена часть HTML-кода с вызовом апплета MY_APPLET (в данном случае броузер выделит для исполнения апплету область экрана размером 200×100 пиксел и присвоит формальному параметру YourBorn апплета значение 1967)

```
. . .
<title>Пример использования апплета в HTML</title>
<body>
<applet code="my_applet.class" width=200 height=100>
<param name=YourBorn value="1967">
</applet>
</body>
. . .
```

В настоящее время язык Java применяется даже при создании серверной и клиентской частей ПО баз данных, возможность выполнения апплетов на различных конфигурациях ЭВМ в Сети породила *аглеты* (комбинация терминов *апплет* и *агент*) - Java-программы, перемещающиеся между узлами Сети и выполняющие специальные (заданные программистом, например, поиск специфической информации) функции (с отс**ы**лкой найденной информации узлу-резиденту). В настоящее время язык Java считают перспективным для обеспечения управления самой разнообразной бытовой техникой через сеть InterNet.

Комплект Java-компилятора и интерпретатора Java Developers Kit можно получить на адресе java.sun.com/JDK-1.0.

Для разработки программ на Java существуют интегрированные ср**е**ды Microsoft Visual J++, Symantec Cafe, JavaWorkShop (*Sun Microsystems*) соответствующее ПО есть и у фирмы Inprise Corp. В 1999 году фирма Sun Microsystems представила (предварительный) вариант спецификации платформы Java2, внедрение которого должно помочь снизить затраты на Java-разработки и повысить совместимость серверных приложений (см. www.gartner.com/webletter/sunus/default.html).

Желающим более подробно ознакомиться с концепциями и языком Java рекомендуются работы [16,19,21] и WEB-сайт компании Sun Microsystems java.sun.com, один из примеров (относительно сложной Java-программы) расположен на spectr.orc.ru.

Существуют альтернативы проекту Java - например, компания General Magic предлагает язык Telescript, позволяющий создавать агенты, ‘перепол-

зающие’ с одного компьютера на другой, сохраняя в памяти обнаруженную информацию (что запрещено Java-апплетам по требованиям безопасности).

Ответом фирмы Microsoft Corp. на разработку фирмой SUN языка Java является (независимая от языка программирования) технология dotNet (.Net) (поддерживаемая, в частности, сетевой сред**о**й исполнения Net Framework, платформой для разработки офисных приложений Office.Net, языком C# и сред**о**й программирования Visual Studio.Net и др., см. www.microsoft.com/net, msdn.microsoft.com/net, msdn.microsoft.com/vstudio/nextgen), являющаяся более богатой, чем Java как по идеям, так и по возможностям реализации. Система .Net является комплектом сетевых служб, позволяющих ПК обращаться к расположенным на WEB-серве**а**х файлам данных и прикладным программам.

Байт-коды Java (как, впрочем, и объекты ActiveX), являясь весьма привлекательными для применения (т.к. позволяют существенно расширить функциональность приложений), потенциально весьма опасны с точки зрения возможности несанкционированного проникновения (и совершения непредсказуемых действий) в любой доступный по Сети компьютер и поэтому требуют особых предосторожностей при использовании.

7.2.3.**ЯЗЫКИ JavaScript, VBScript и PerlScript**

При всей своей универсальности язык Java достаточно сложен в использовании; во многих случаях не требуется мощности Java, однако желательно производить некоторые действия (например, арифметические, недоступные HTML).

С этой целью был разработан (фирма Netscape Communication Corp., первоначальное название LiveScript) язык JavaScript (не имеющий прямого отношения к Java), язык JavaScript является средством создания активных WEB-страниц непрофессионалами. Фирма Microsoft Corp. реализовала в своем броузере Microsoft Internet Explorer поддержку языка JScript (весьма близкому к JavaScript), давняя приверженность главы Microsoft к Basic’у привела к включению в упомянутый броузер поддержку языка VBScript (по возможностям близкого JavaScript). Так же как и Java, программы на JavaScript и VBScript позволяют не только создавать активные (изменяющиеся предсказанным образом в процессе работы) WEB-страницы, но и снижают поток данных между рабочей станцией и сервером WEB (что благоприятно сказывается на скорости реакции сервера).

Ниже показан HTML-сценарий с встроенным кодом на JavaScript

```
<HTML>
<HEAD>
<TITLE>Динамическое создание WEB-страницы</TITLE>
</HEAD>
```

```
<BODY>
<H1>JavaScript test</H1>
<SCRIPT LANGUAGE="JavaScript">
   document.write("Этот текст динамически сгенерирован "  +
                  "программой на JavaScript");
</SCRIPT>
</BODY>
</HTML>
```

Здесь код на JavaScript заключен между тегами <SCRIPT LANGUAGE="JavaScript"> и </SCRIPT> и вызывает метод **write** объекта **document**, причем метод (по умолчанию) вызывается при загрузке HTML-файла броузером, что приводит к генерации соответствующего текста (и, соответственно, индикации его в окне броузера).

Язык JavaScript позволяет связать вызов конкретной функции с некоторым событием (загрузкой или выгрузкой файла, перемещением указателя 'мыши' над заданным участком окна броузера, щелчком кнопки 'мыши' над графическим объектом и др.), что позволяет придать WEB-странице определенную 'интеллектуальность'.

Например, следующий фрагмент HTML-сценария связывает вызов JavaScript-процедуры **MakeOnLoad** с загрузкой файла в броузер, а процедуры **MakeOnUnload** - с выгрузкой (например, в связи с переходом к следующей странице).

```
. . .
<BODY onLoad="MakeOnLoad()" onUnload="MakeOnUnload()">
. . .
```

Следующий пример сценария служит для выдачи окна запроса с двумя кнопками - 'Ok' и 'Cancel' (при щелчке левой кнопкой 'мыши' по выделенному словосочетанию-ссылке 'Желаете выгрузить') и совершения некоторого действия (в данном случае выгрузки файла OMD.RAR на компьютер клиента) при положительном ответе

```
. . .
<A HREF="JavaScript:
if (confirm('Вы действительно желаете выгрузить на свой '  +
            'компьютер файл OMD.RAR размером аж 330 kb ?'))
   location.href = './bin/omd.rar';">
Желаете выгрузить</A> на Ваш компьютер пакет OMD ?
. . .
```

В языке JavaScript определены функции работы с числами, строками, массивами, определения типа броузера, анализа и изменения содержимого WEB-страниц, диалога с пользователем, обработки данных форм (перед от-

сылкой на сервер), взаимодействия с апплетами Java и др., поддерживается обьектно-ориентированный подход. Для отладки JavaScript-сценариев предлагается отладчик Microsoft Script Debugger.

Язык VBScript в целом обладает сходными с JavaScript возможностями, но имеет Basic-подобный синтаксис и поддерживается только броузером Microsoft Internet Explorer.

Желающим более подробно ознакомиться с концепциями и языком JavaScript рекомендуются работы [18,19], некоторые (несложные) конструкции JavaScript можно выгрузить (в составе HTML-сценариев) с сайта автора pilger.mgapi.edu. Более серьезные примеры JavaScript-приложений можно получить с адресов

- www.cris.com/raydaly/hjdemo.shtml
- www.homepages.com/fun/I040EZ.html
- www.geocities.com/SiliconValley/7116/jv_cale.html
- www.best.com/nessus/jstodasy.html

В отличие от Java, коды JavaScript и VBScript практически безопасны с точки зрения совершения несанкционированных действий на компьютере пользователя.

Некоторые WEB-броузеры (например, Microsoft Internet Explorer версий выше 4) способны интерпретировать встроенный в HTML-код язык PerlScript. Часто вместо создания полномасштабной CGI-программы возможно ограничиться внедрением текста на PerlScript в WEB-страницу (ниже приведен пример простейший HTML-страницы с выводом строки посредством PerlScript-кода), также см. [20].

```
<!DOCTYPE HTML PUBLIC “-//W#C//DTD HTML 3.2//EN”>
<HTML>
<HEAD>
<TITLE>My first PerlScript  example</TITLE>
</HEAD>
<BODY>
<H2>PerlScript example</H2>
<SCRIPT LANGUAGE="PerlScript">
$window->document->write("Hello, PERL !");
</SCRIPT>
</BODY>
</HTML>
```

7.2.4.СЕРВЕРНЫЕ РАСШИРЕНИЯ CGI И ISAPI

Серверные расширения CGI (*Common Gateway Interface* - стандартный шлюзовый интерфейс) и ISAPI (*Internet Server API*, также NSAPI - *Netscape*

Server API) предназначены для запуска внешних программ под управление WEB-сервера; внешняя программа получает информацию через протокол HTTP от удаленного пользователя, обрабатывают ее (например, осуществляют запрос к базе данных) и возвращают результат обработки обратно в виде ссылки на существующий HTML-документ или в виде динамически созданной HTML-страницы.

Передача информации от удаленного пользователя происходит следующим образом

- В создаваемом для ввода информации документе HTML размещается *форма ввода*, состоящая из необходимых органов управления (полей редактирования текстовой информации, переключателей, списков и др., каждому органу управления присваивается произвольное имя; в форме должна присутствовать кнопка, нажатие которой инициирует передачу информации из полей формы на сервер).
- Данные поступают на сервер и обрабатываются (возможно, весьма изощренными) приложениями CGI или ISAPI.
- CGI/ISAPI-приложение генерирует (обычно динамически) HTML-документ (файл) и пересылает его обратно удаленному пользователю (где этот документ интерпретируется и визуализируется броузером).

Ниже приведен пример HTML-кода простой формы с двумя полями ввода (имена полей ‘text1’ и ‘text2’) и кнопкой для отс**ы**лки сообщений серверу

```
. . .
Образец простейшей формы
<FORM METHOD=GET ACTION=“http://www.my_server.ru/cgi/form.exe”>
<TABLE border=0>
<TR>
<TD>Имя :</TD>
<TD><INPUT TYPE=text NAME="text1" VALUE="Текст 1" size=33>
</TD></TR>
<TR>
<TD>Пароль : </TD>
<TD><INPUT TYPE=text NAME="text2" VALUE="Текст 2" size=33>
</TD></TR>
<TR>
<TD> </TD>
<TD><INPUT TYPE=submit VALUE="Послать данные серверу">
</TD></TR>
</TABLE>
</FORM>
. . .
```

На рис.7.12 показано отображение этой формы броузером (хорошо видны два поля ввода и кнопка отс**ы**лки введенных в эти поля данных на сервер).

Параметр ACTION описания формы определяет действие, выполняющееся над присланной на сервер информацией (в данном случае указан путь к программе CGI, которая будет выполнять обработку данных). Параметр METHOD выбирается один из двух методов передачи данных серверу WWW - при значении этого параметра GET указанная в параметре ACTION программа CGI получит данные из формы через переменную среды с именем QUERY_STRING, в случае METHOD=POST программа CGI получит данные из формы через стандартный поток ввода **stdin**.

С целью использования языков программирования, не поддерживающих (в явном виде) стандартных потоков ввода и вывода (например, Pascal) разработана спецификация WinCGI, согласно которой в передаче данных используются привычные для Windows инициализационные файлы [6].

Возможна прямая пос**ы**лка серверу строки query-string в соответствие со следующим URL (через знак вопроса после имени обрабатывающей запрос CGI-программы указывается пересылаемая строка)

http://www.my_server.ru/cgi/search.exe?query-string

При использовании METHOD=GET данные формы поступают на сервер в виде значения переменной среды QUERY_STRING в следующем формате:

Имя1=Значение1&Имя2=Значение2&Имя3=Значение3

Здесь в качестве имен используются значения параметров NAME формы, вместо значений подставляются данные из соответствующих именам полей. Программа CGI должна просканировать содержимое текстовой строки переменной сред**ы** QUERY_STRING и по имени поля найти нужное значение, введенное в это поле пользователем. Адрес заданной строки переменной среды в программе легко получить с помощью С-функции **getenv**

```
char * szQueryString;
szQueryString = getenv("QUERY_STRING");
```

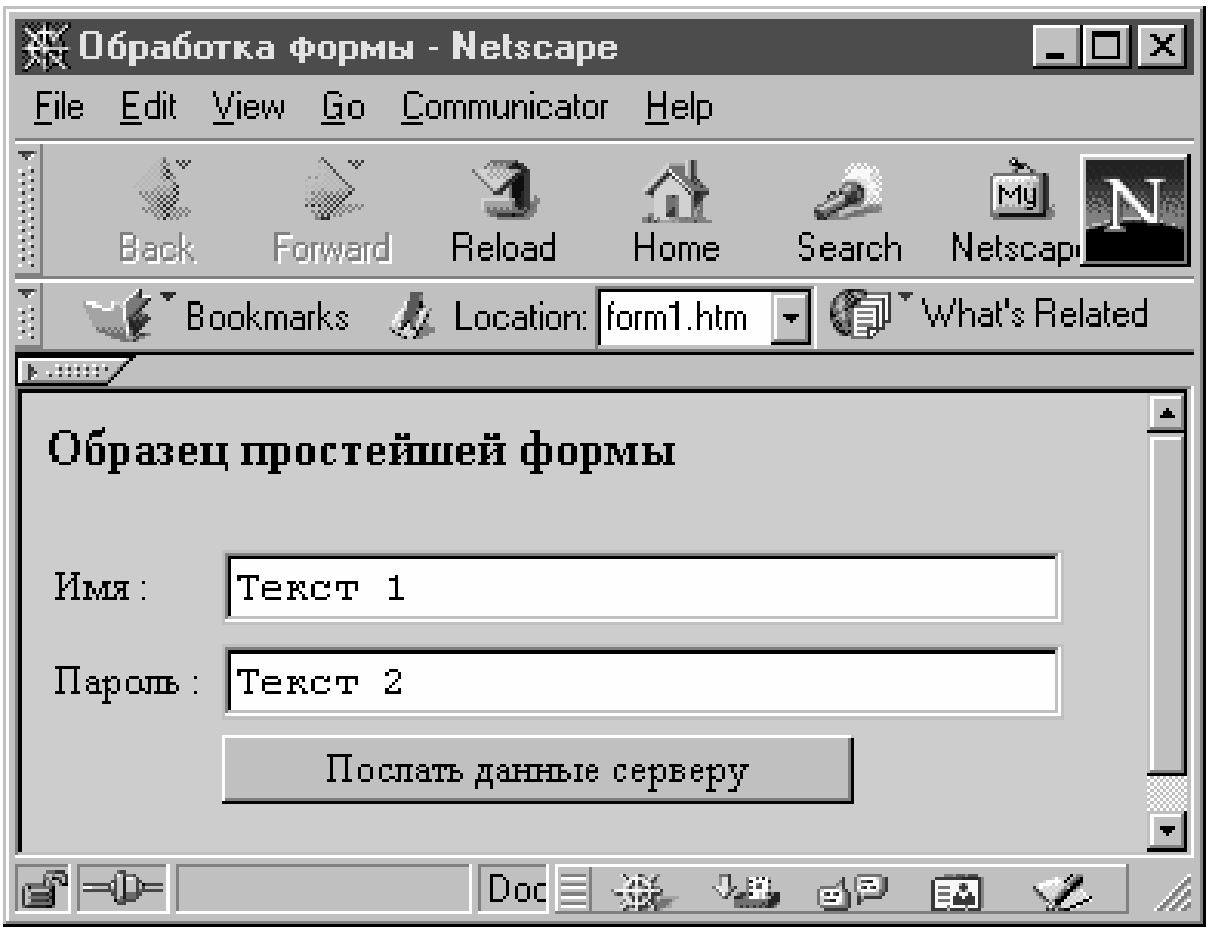

Рис.7.12. Отображение простейшей формы броузером.

Передаваемая в переменной среды QUERY_STRING строка закодирована с использованием т.н. кодировки URL (символы пробелов заменяются на символ ‘+’ , для представления кодов управляющих и некоторых других символов используется конструкция вида **%xx**, где **xx** - шестнадцатиричный код символа в виде двух ASCII-символов); CGI-программа должна выполнить обратную перекодировку.

При использовании METHOD=POST программа CGI получает данные из формы через стандартный поток ввода **stdin** (для чтения удобно использовать С-функции **fread** или **scanf**) в аналогичном методу GET формате, причем количество байт в **stdin** передается CGI-программе через переменную среды с именем CONTENT_LENGTH

```
int Size;
Size = atoi(getenv("CONTENT_LENGTH"));  //получить длину строки

char szBuf[8196];
fread(szBuf, Size, 1, stdin);  //прочитать полученные данные
```

Метод GET применяется относительно редко (длина строки QUERY_STRING ограничена), метод POST более предпочтителен.

Вне зависимости от примененного метода передачи данных (GET или POST) результат своей работы программа CGI должна направлять в стандартный поток вывода **stdout** (при этом WWW-сервер гарантирует возврат данных броузеру локального компьютера). Ниже приведен пример динамической генерации CGI-программой документа HTML и направления его на **stdout**

```
. . .
printf("Content-type: text/html\n\n");   //упрощенный пролог HTML
printf("<!DOCTYPE HTML PUBLIC \""-//W3C//DTD HTML 3.2//EN\"">");
printf("<HTML><HEAD>
<TITLE>My first dinamic HTML-page</TITLE>
</HEAD><BODY>");
printf("<H1>Результаты обработки данных формы:</H1>");
. . .
. . .
printf("</BODY></HTML>");
```

После получения возвращаемых HTML-данных они интерпретируются броузером. При нежелании генерировать сложный HTML-код подобным (весьма кропотливым) образом можно применить метод создания шаблона HTML-файла с последующей программной его модификацией и записью в **stdout**.

При использовании CGI-программ через (другие) переменные среды (а их несколько десятков) передается большое количество очень важной информации, которая может существенно помочь программисту.

Заметим, что с помощью CGI-программ легко реализуются, например, счетчики числа посещений страниц; в качестве языка программирования CGI-программ часто используют интерпретатор языка Perl *(Practical Extraction and Report Language,* иногда в шутку *Pathologically Eclectic Rubbish Lister,* см. InterNet-адреса www.perl.com, www.cpan.org, orwant.www.media.mit.edu/the_perl_journal, www.tpj.com) [20], хотя может быть применен практически любой язык программирования (особенно удобен C/C++). Например, в работе [16] приведены исходные тексты CGI-приложений на языке Java, в [20] - на языке Perl; на WEB-сайте автора данной работы pilger.mgapi.edu реализована простая гостевая книга с применением Perl.

Недостатком приложений CGI является то, что для обработки каждого запроса WWW-сервером он запускает новое CGI-приложение (новый процесс), а т.к. современные сервер**а** одновременно могут обрабатывать много запросов, ресурсы машины-сервера (например, объем оперативной памяти) быстро истощаются (не говоря уже о снижении производительности серве-

ра). От подобного недостатка свободен метод ISAPI, основанный на обработке запросов динамически загружаемыми (из DLL-библиотек) функциями (недостаток - т.к. ISAPI-расширение выполняется в том же адресном пространстве, что и сам WEB-сервер, критическая ошибка ISAPI-приложения обычно вызывает крах сервера). В отличие от CGI, ISAPI-приложение получает данные не из стандартного потока, а с помощью специально предназначенной для этого функции интерфейса ISAPI; вместо стандартного потока вывода также применяется специальная функция [15].

Некоторые сервер**а** (например, Apache) содержат *встроенный* Perl, при этом для каждого поступающего CGI-запроса сервер создает новый поток (вместо нового процесса); это значительно ускоряет выполнение CGI-запросов указанным сервером. Для работы под Windows'NT разработаны (см. www.activeware.com) пакеты PerlScript (разработка сценариев ActiveX) и PerlIS (динамически вызываемая библиотека ISAPI-интерфейса [20].

Известны как *расширения ISAPI* (по функциям аналогичны CGI-расширениям сервера), так и *фильтры ISAPI* (фактически являющиеся *брандмауэрами,* применяются для шифрования или перекодировки проходящих через сервер данных, компрессии информации, сбора статистических данных о пользователях, проверки прав д**о**ступа и др.).

При использования метода ISAPI имя соответствующего DLL-файла описывается в параметре ACTION формы (аналогично CGI), также описывается параметр METHOD, однако сами присланные на сервер данные могут быть получены с помощью специально зарезервированных функций **GetServerVariable** (чтение значений переменных среды) и **ReadClient** (собственно чтение присланных данных), отс**ы**лка же данных (как и при использовании CGI, обычно в виде динамически создаваемых HTML-файлов) производится функциями **WriteClient** и **ServerSupportFunction**.

Желающим более подробно ознакомиться с возможностями серверных расширений CGI и ISAPI рекомендуются работы [6,15,19].

Заметим, что CGI- и ISAPI-программы потенциально являются источниками снижения безопасности функционирования WEB-сервер**о**в. Например, можно советовать работу [20] для ознакомления с началами обеспечения безопасности выполнения CGI-сценариев на Perl; ниже приведено несколько WEB-адресов, посвященных проблеме безопасности

- www.w3.org/Security/Faq/www-security-faq.html
- www.perl/com/CPAN-local/doc/FAQs/cgi/perl-cgi-faq.html
- stars.com/Authoring/Scripting/Sequrity
- www.go2net.com/people/paulp/cgi-sequrity/safe-cgi.txt

Мощным средством для упрощения процесса реализации функциональности и расширения возможностей технологии CGI является язык PHP/FI

(*Персональные инструментальные средства для Домашней Страницы / Интерпретатор Форм*), предложения которого встраиваются непосредственно в текст HTML-страницы и выполняются процессом, инициализированным сервером (обычно Apache). PHP/FI существенно упрощает обработку запросов от форм и анализ SQL-запросов, допускает добавление пользовательских функций (обычно написанных на C). Использование PHP/FI повышает эффективность обработки запросов (CGI-программа не стартует, PHP/FI-код выполняется одним из серверных процессов), при этом повышается уровень защиты данных и конфигурируемость серверного ПО. Первым признаком того, что страница обрабатывается PHP/FI, является добавление нижнего колонтитула с информацией о количестве обращений к данной странице (если программа скомпилирована с опцией регистрации д**о**ступа).

7.3.**ПОИСК ИНФОРМАЦИИ В СЕТИ InterNet**

Необходимость разработки *справочно-поиск**о**вых систем* в Сети вызвана следующими причинами

- Огромным объемом информации в Сети (по оценкам на начало 2000 года - около 2×10^9 уникальных страниц с удвоением ежегодно, причем ежедневно число страниц увеличивается на 7×10^6).
- Недостаточной (а в некоторых случаях, наоборот, чересчур подробной) структуризацией этой информации.
- Широким тематическим профилем информационных массивов.

Существует две разновидности ресурсов для поиска. Это так называемые *каталоги* (directories) и *поиск**о**вые машины* (search engines).

Сетевые *каталоги* организованы примерно так же, как и библиотечные. Они содержат различные разделы, подразделы и т.д., то есть имеют иерархическую структуру. Работают с этими каталогами так же, как и с библиотечными - ‘спускаясь’ вниз по иерархической лестнице.

Каталоги создаются вручную, т.е. информация в них заносится людьми. Благодаря ‘человеческому фактору’ информация в каталогах организована достаточно четко, что позволяет в определенных случаях достичь требуемого результата быстрее, чем при помощи поисковых машин. С другой стороны, в каталоги попадают далеко не все существующие страницы, а лишь ‘лучшие’ (с субъективной точки зрения создающего каталог). По этой причине найти какую-то достаточно специфическую информацию в каталоге зачастую невозможно.

Наряду с каталогами используются *поисковые машины*. Суть этих механизмов заключается в том, что доступные в Сети страницы автоматически индексируются, т.е. создаются специальные базы данных, содержащие клю-

чевые слова и связанные с ними адреса страниц, а уже в этих базах данных проводится поиск. Таким образом, поисковые машины состоят из программ, собирающих информацию для базы данных, собственно базы, и программ для поиска в этой базе данных.

Индексируют информацию так называемые *роботы* (crawlers, spiders и пр.) - специальные программы, которые 'ползают' по сети, просматривают файлы и создают индексы, причем весь процесс происходит автоматически (разработчик WEB-сайта может ограничивать деятельность роботов с помощью файла ROBOTS.TXT). *Полнотекстовые* поисковые машины индексируют каждое слово на WEB-странице (исключая лишь некоторые зарезервированные слова), *абстрактные* поисковые машины создают некий экстракт каждой страницы.

Кроме этого, существуют и так называемые *мета-системы*, представляющие собой интерфейсы для одновременного поиска с помощью нескольких поисковых машин. Так же многие поисковые системы содержат не только интерфейс для работы с индексом, но и каталоги.

К наиболее известным поисковым системам относятся AltaVista (разработка фирмы *DEC*, рис.7.13), Яndex (*CompTek International*), RAMBLER (*Stack Ltd.*, ориентирована на русскоязычный InterNet, рис.7.14), каталог YAHOO, синтез каталога и поисковой машины Lycos и др. Всего в Сети имеется около 600 систем поиска (см. www.beaucoup.com/engines.html), д**о**ступ к некоторым наиболее известным поисковым системам возможен по адресу pilger.mgapi.edu/right_7.htm#lab_4).

При просмотре сайтов как раз и происходит накопление *ключевых слов*. Многие поисковые системы учитывают информацию из тега **<meta name=keywords content=...>** (заполняемого разработчиком WEB-страницы) при создании массива ключевых слов. Поисковая машина RAMBLER, например, игнорирует содержимое этого тега и осуществляет сканирование HTML-текста в поисках ключевых слов с учетом расположения и частот**ы** их встречаемости, разработчики RAMBLER'а считают, что такой подход повышает *релевантность* (уровень соответствия между текстом запроса и документа, к которому этот запрос направлен) при поиске по ключевым словам.

Таким образом, поисковые системы постоянно накапливают информацию о ресурсах сети InterNet и используют ее при запросах на поиск.

Применяются следующие модели индексирования и поиска - *векторная модель информационного потока,* основанная на теории *нечетких множеств* модель*, вероятностная модель*. Используются следующие типы *информационно поисковых языков* (ИПЯ) - *традиционные ИПЯ, системы взвешивания терминов, ИПЯ типа 'Like this'*. В качестве способов коррекции результатов поиска применяются методы *фильтрации, коррекции по релевантности, кластеризации.* Используются следующие механизмы улучшения запроса - *про-*

стой запрос, сложный запрос, нормализация лексики, ранжирование, коррекция по релевантности.

Рис.7.13. Главное окно поисковой системы AltaVista.

Каждая поисковая система имеет строку для ввода *запроса на поиск*, в которую пользователь вводит ключевые слова для поиска. В большинстве случаев пользователь имеет возможность вводить несколько ключевых слов (допускаются символы типа * для указания любого набора символов и др.), связывая их логическими отношениями типа OR, AND, NOT и др., например, возможна строка для поиска вида (в реальности указанные логические связки часто представляются иными символами)

((микросхемы OR чипы) AND память

Некоторые поисковые системы (например, Яndex) содержат алгоритмы морфологического анализа и синтеза, основанные на базовом словаре, умеют нормализовать слова, то есть находить их начальную форму, а также строить гипотезы для слов, не содержащихся в базовом словаре.

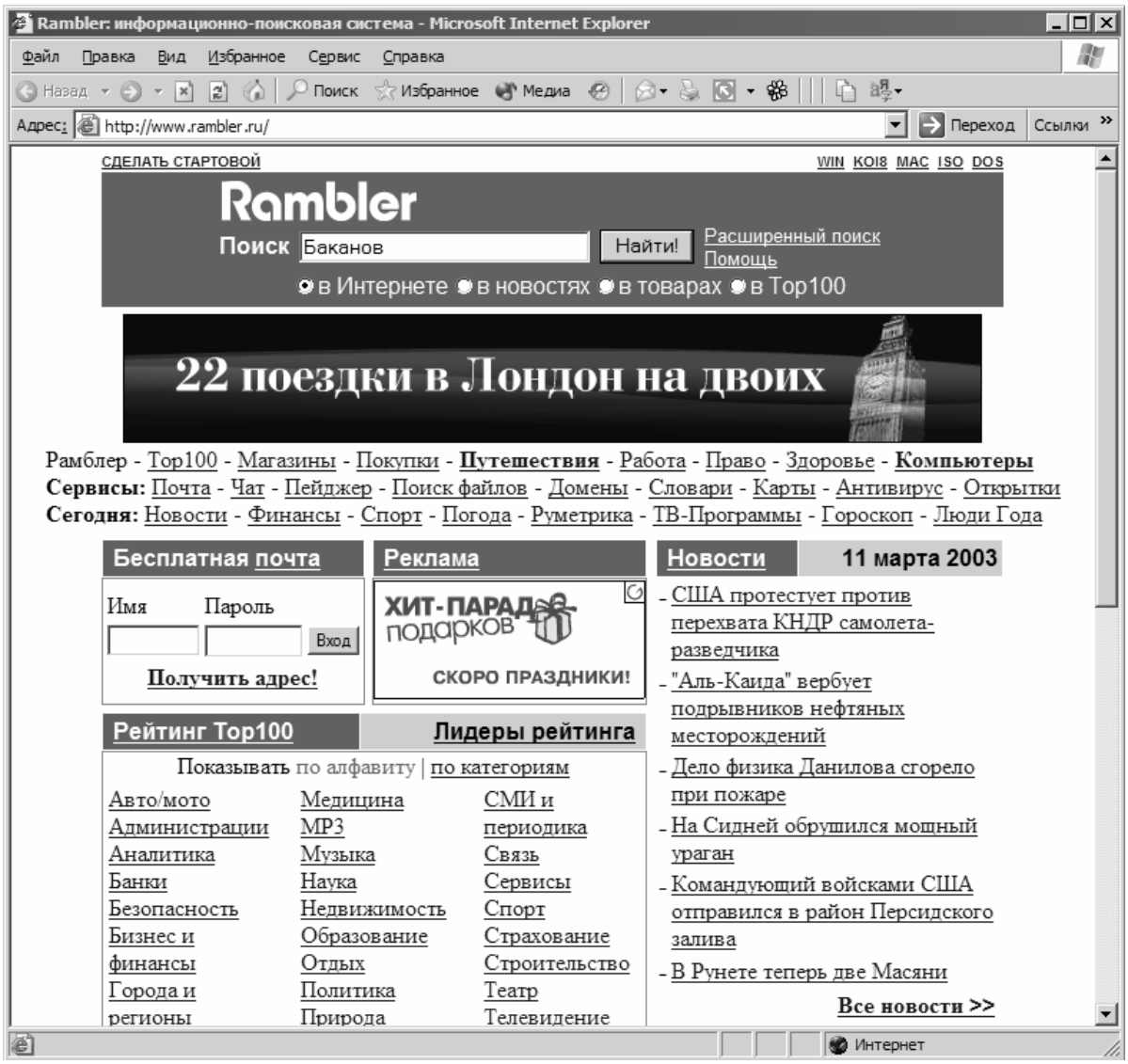

Рис.7.14. Главное окно поиск**о**вой системы RAMBLER.

Многие механизмы поиска предусматривают еще одну возможность - указание расстояния между терминами в документе. Таким образом, можно отбрасывать длинные файлы, в которых одно слово используется в разных контекстах. Например, с помощью оператора NEAR указывается, что второй термин должен находиться на расстоянии, не превышающем определенного числа слов. FOLLOWED BY используется для получения документов, в которых искомые термины встречаются в заданном порядке, а ADJ применяется для поиска смежных (следующих друг за другом) терминов.

Одной из проблем при поиске информации в Сети является выдача слишком большого объема информации на введенный запрос (запрос слишком широк, т.е. неконкретен), в этом случае можно воспользоваться методом уточнения запроса (например, зад**а**в более жесткие условия поиска)

((микросхемы OR чипы) AND память AND постоянная

В настоящее время программное обеспечение поиска информации в Сети является, пожалуй, наиболее динамично развивающейся областью (как и *теория информационно-поисковых систем*).

7.4. **РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ InterNet**

Использование сред**ы** программирования Microsoft Visual C++ и библиотеки классов MFC (*Microsoft Foundation Classes*) позволяет создавать программное обеспечение для InterNet и INTRANET для операционных систем серии Windows.

Фирмой разработан программный интерфейс Win32 Internet (называемый также WinInet, соответствующие функции расположены в файле WININET.DLL, описание функций можно получить на адресе www.microsoft.com/win32dev) специально для создания работающих с протоколами HTTP, FTP и GOPHER приложений (при этом программист избавлен даже от необходимости программирования на уровне сокетов).

Например, несложно создать упрощенный вариант WEB-броузера. Дело в том, что основу броузера Microsoft Internet Explorer составляют несколько DLL-библиотек, в которых определены объекты ActiveX; таким образом фирма Microsoft Corp. предоставляет средства встраивания компонентов своего броузера в любое пользовательское приложение (путем использования органа управления Microsoft Web Browser Control из сред**ы** программирования Visual C++ или вызова функций из DLL-библиотек, где определены соответствующие объекты ActiveX). Средства сред**ы** Visual C++ позволяют загружать ресурсы из InterNet (входящая в состав API Windows функция **ShellExecute** расширена до возможностей работы с удаленными файлами по Сети, подробнее см. информацию по адресу dials.ccas.ru/frolov/rwin/webhelp.html и работу [17], там же приведено большое количество исходных текстов на C++).

Другой показательный пример (собственноручной) разработки приложений для InterNet - управление броузерами с помощью стандартных для Windows технологий *динамического обмена данными* (DDE, *Dinamic Data Exchange*) и *связывания и внедрения объектов* (OLE, *Object Linking and Embedding*). В самом деле, в большинстве случаев нет смысла создавать (новый) броузер, целесообразнее уметь использовать имеющиеся разработки в собственных целях. Фирма Netscape Communications Corp. сделала свои броузер и сервер пригодными для функционального расширения, опубликовав свойственные им API-функции.

С использование DDE появляется возможность, например, управлять загрузкой URL и позиционировать окно броузера Netscape посредством ко-

манд пользовательской программы; в общем случае DDE-интерфейс позволяет управлять многими физическими аспектами функционирования броузера. Возможности OLE дают возможность пользовательским программам использовать возможности броузера (практически полный импорт функций броузера в пользовательское приложение). Большой объем справочных данных и исходных текстов на Pascal'е для реализации подобных приложений приведен в работе [6].

В 1999 г. фирма Netscape Communications Corp. объявила об опубликовании исходных кодов своего броузера (с целью их модификации в направлении оптимизации и развития).

В самые последние годы сеть InterNet стала ареной для осуществления грандиозных проектов. Пожалуй, наиболее масштабным проектом по распределенной обработке данных является проект SETI@home, представляющий собой выполнение на сотнях тысяч компьютеров добровольцев всего мира специальной программы обработки результатов сканирования неба радиотелескопами с целью поиска сигналов разумной жизни (фрагменты программы выполняются в моменты пауз в работе подключенных к Сети компьютеров, на данный момент времени объем обработанной информации эквивалентен 90×10^3 лет работы мощного процессора). Об общем объеме информации, необходимой для полного выполнения расчетов по проекту SETI@home говорит следующий факт - необходимый для обработки достаточно малой порции данных ресурс составляет 175×10^9 операций (около 25 часов непрерывной работы персонального компьютера с процессором класса i586); более подробную информацию о проекте SETI@home можно получить на WEB-адресе setiathome.ssl.berkeley.edu/about_seti/about_seti_at_home_1.html (русскоязычное зеркало setiathome.spb.ru). Интересен проект 'Техносфера' (www.technosphere.org.uk), представляющий сложный симулятор для моделирования взаимодействия тысяч персонажей на едином поле жизни. Изложенные факты говорят об огромных (на данное время часто даже трудновообразимых) возможностях сети InterNet.

8.**ЗАЩИТА ИНФОРМАЦИИ В КОМПЬЮТЕРНЫХ СЕТЯХ**

8.1.**БЕЗОПАСНОСТЬ СЕТЕЙ**

В связи с широким использованием сетей ЭВМ государственными, военными и финансовыми структурами большое внимание уделяется вопросам защиты конфиденциальной информации, передаваемой по сети, от несанкционированного прослушивания; особенно это важно при работе в открытой любому пользователю сети InterNet; этому служит специальное программное обеспечение.

Безопасность любой сети включает три аспекта

- Физический - предупреждение физического д**о**ступа к аппаратному обеспечению (например, наличие закрываемого на замок компьютерного помещения или склада).
- Процедурный - действия, выполняемые пользователями компьютера для повышения безопасности используемых процедур.
- Логический - меры по обеспечению безопасности программного обеспечения (например, защита информации паролем, шифрование информации).

Хотя указанные три аспекта безопасности всегда должны выполняться в комплексе (если не учитывать любой из них, вся система защиты разрушается), в аспекте данной работы представляет интерес третий (логический, достигающийся в большинстве случаев программным путем).

Один из вариантов защиты на логическом уровне - использование *брандмауэра (firewall)* - отдельной точки контакта между частной и общедоступной сетью; функции брандмауэра обычно выполняет сервер с соответствующим программным обеспечением, выполняющим функции защиты и/или фильтрации передаваемых данных (один из вариантов брандмауэра - *proxy-сервер*, используемый для выхода в InterNet с локальной сети). Функции брандмауэра может выполнять ПО, выполненное в виде фильтра ISAPI, способного контролировать весь проходящий через сервер поток данных [15].

Наиболее надежный метод защиты информации в компьютерных сетях - метод шифрования сообщений [6]. Шифрованием называют процесс преобразования сообщения, при котором оно может быть восстановлено в исходной читабельной форме только тем получателем, для которого оно предназначено.

С помощью случайного кода можно достаточно просто выполнить шифрование сообщения, однако из-за непредсказуемости характера такого процесса шифрования никто (даже сам автор сообщения) не сможет расшифровать это сообщение. Надежная схема должна использовать *регулярный код* для шифрования, позволяющий получателю быстро расшифровать сообщение.

Указанные коды называются *ключами (keys)*. Размер и тип ключа (в общем случае ключи являются битовыми кодами) определяет, насколько трудно будет раскрыть код шифровки и расшифровать сообщение. Например, 8-битовый код имеет 256 различных комбинаций, 40-битовый имеет 2^{40} = 1’099’511’627’776 возможных комбинаций, расшифровка 128-битового ключа практически невозможна методом перебора (по некоторым оценкам, для такой расшифровки необходимо наличие $4{,}2\times10^{22}$ процессоров производительности 256 млн. операций шифрования в секунду, в этом случае ключ бу-

дет взломан за год; стоимость такого количества процессоров в 2000 г. оценивается в $3{,}5 \times 10^{24}$ US$). Содержащаяся в ключе информация используется в дальнейшем для шифрации (методом преобразования битов в потоке) и дешифрации передаваемых по сети сообщений, причем на собственно операцию шифрования приходится небольшая часть уровня секретности (т.о. можно считать сообщение практически рассекреченным, если известен ключ шифрования).

Имеется два типа ключей - *симметричный* и *асимметричный*.

Симметричный ключ - ключ, в котором отправитель и получатель используют один и тот же ключ для шифрования и дешифрования сообщений. Использование одного ключа является недостатком схем шифрования с симметричным ключом - ведь сам код ключа должен передаваться незашифрованным, чтобы получатель мог использовать его для дешифрации сообщений. Если постороннее лицо получит такой код ключа, он может расшифровать сообщение независимо от того, сколько битов составляет длина ключа. Правительство США определило и утвердило *стандарт шифрования данных (DES*, *Data Encryption Standart)*, который представляет собой схему шифрования с симметричным секретным ключом, стандарт DES работает на 64-битовых блоках посредством 56-битового ключа.

Схемы шифрования с помощью асимметричного ключа используют *общедоступный* и *частный* ключи. Тот, кто желает получить зашифрованное сообщение, должен иметь оба ключа. Получатель предоставляет общедоступный ключ, а отправитель затем использует его для шифрования сообщения. Единственным способом дешифрования этого сообщения является использование обоих - общедоступного и частного ключа получателя. Даже отправитель не сможет дешифровать это сообщение, поскольку он не знает частного ключа получателя.

Шифрование RSA - алгоритм для асимметричного шифрования с помощью общедоступного ключа, запатентованный в 1983 году фирмой Public Key Partners (PCP); символы RSA суть аббревиатура фамилий изобретателей алгоритма. RSA в настоящее фактически стандарт *de-facto* и является скорее дополнением, чем заменой шифрования DES. Каждая схема имеет свои преимущества - DES является быстродействующей схемой шифрования и работает эффективно для крупных файлов (имеет лучшие скоростн**ы**е качества), RSA работает эффективно для шифрования относительно небольших сообщений, обеспечивает цифровые подписи и надежный обмен ключами, не требуя при этом предварительного обмена секретными кодами.

Аутентификация - процесс, который убеждает получателя документа в подлинности отправителя и в целостности документа. Этот процесс выдает цифровую подпись, созданную из *дайджеста сообщения* (*message digest*), представляющего собой уникальную строку битов, основанную на содержимом сообщения. Подпись является неподдельной строкой данных, которая

аутентифицирует, что конкретное лицо создало содержимое документа и согласно с ним. Отправитель создает дайджест сообщения и шифрует его с помощью частного ключа, дайджест сообщения сопровождает это сообщение. Получатель расшифровывает дайджест сообщения и сопоставляет его с самим сообщением. Если они совпадают, процесс аутентификации является достоверным (можно шифровать и подписывать сообщения одновременно).

Существует другой процесс аутентификации, использующий цифровые сертификаты; эти сертификаты содержат следующее

- Общедоступный ключ.
- Отличительное имя (информация об имени и адресе).
- Дату выдачи и срок хранения.
- Цифровую подпись удостоверяющей организации.

Удостоверяющей организацией является CA (Certifying Authority), например, VerySign, которая создает цифровые сертификаты; CA обычно взимает плату за эту услугу и публикует свой общедоступный ключ и отличительное имя (Distinguished Name) для того, чтобы другие пользователи могли добавить их в броузеры и WEB-серверы в качестве элемента их доверенного каталога (который недоступен для зарегистрированных на сервере пользователей).

Алгоритм RSA запускается с помощью *двух больших простых чисел* P и Q, числа P и Q дают N - модуль (остаток деления P на Q); существует документация в отношении RSA, где рекомендуется выбирать пару ключей с *сильными (strong)* простыми числами. Сильные простые числа - такие числа, которые имеют свойства, делающие их модуль N трудным для факторизации (процесса разбиения целого числа на набор целых чисел - *факторов* - которые после умножения дают исходное целое число). Дело в том, что умножение двух простых чисел выполняется достаточно просто, однако факторизация (вышеуказанное разбиение целого числа) гораздо труднее. Такой процесс именуется *односторонней функцией (one-way function)*, которую легко выполнять в одном направлении, но гораздо труднее - в обратном.

Еще одна проблема при выборе простых чисел связана с размером модуля - большой модуль создает более надежную схему шифрования, однако он замедляет процесс шифрования. Например, для факторизации 512-битового модуля (при этом каждое из простых чисел P и Q должно иметь длину около 256 битов) требуется усилие, эквивалентное $8{,}2\times10^{6}$ $US.

После определения модуля выбирается число E, м**е**ньшее, чем N и также б**о**льшее по сравнению с произведением (P-1)(Q-1), находится его модуль D. Общедоступной парой ключей является (N,E), частным ключом - D (следует хранить в секрете числа P и Q).

Зная значения P и Q, любой пользователь может восстановить зашифрованную информацию. Лучше всего получать случайные значения P и Q с помощью физического процесса (перемещения ‘мыши’, нажатия клавиатуры, многие связанные с Банком Федерального Резерва США американские банки используют основанную на специальной периферийной карте систему FedWire II).

Дополнительную информацию о защите данных по алгоритму RSA можно получить на WEB-адресе www.rsasecurity.com.

Один из разработчиков RSA, Ron Rivest, предложил более мощный алгоритм шифрования, получивший название RC4. Правительство США запрещает экспорт любых программно-аппаратных реализаций алгоритма шифрования RC4 с длиной ключа более 64 бита без специальной лицензии, приравнивая его к экспорту оружия.

В СССР в период ‘холодной войны’ был разработан алгоритм ГОСТ 28147-89 с длиной ключа 256 бит для защиты составляющих государственную тайну сведений, практически не накладывающий ограничений на степень секретности защищаемой информации.

8.2.**СПЕЦИФИКАЦИИ БЕЗОПАСНОСТИ**

Спецификации безопасности представляют собой рекомендации по реализации концепции безопасности, основанной на имеющихся теориях. Ниже (кратко) рассматриваются спецификации SSL, PCT и STT и др.

Уровень безопасности гнезда (SSL, *Secure Socket Layer*) представляет собой технологию защиты InterNet общего пользования, разработанной фирмой Netscape Communications Corp. и применяемой в продуктах фирмы - броузере Netscape Navigator и сервере Netscape Commerce Server. SSL является многоуровневым протоколом, реализует алгоритм шифрования RSA для защиты частной и аутентифицированной связи в InterNet и осуществляет уплотнение записей с помощью определяемого в текущем сеансов алгоритма сжатия. SSL *не зависит от протокола приложения*; протокол более высокого уровня может располагаться поверх протокола SSL, однако протокол SSL требует надежного протокола передачи (такого как TCP). Функции интерфейса **Windows Socket** ver.2.0 предлагают определенную поддержку SSL.

Технология частной связи (PCT, *Private Communication Technology)* разработана фирмой Microsoft для предотвращения электронного подслушивания в приложениях ‘клиент-сервер’; протокол совместим с SSL и устраняет некоторые слабые места последнего. PCT также независим от протокола приложения, поэтому протоколы более высокого уровня (HTTP и FTP) могут располагаться выше его.

Протокол *технологии безопасных транзакций* (STT, *Secure Transaction Technology*) является совместной разработкой фирм Visa и Microsoft по за-

щите транзакций с применением банковских карточек по открытым сетям типа InterNet. В целях предотвращения обмана при произведении финансовых операциях в сети InterNet в рамках протокола STT решаются следующие задачи

- Предоставление конфиденциальности информации о платежах.
- Аутентификация владельцев карточек и торговцев.
- Обеспечение целостности данных о платежах.

Фирмы Master Card, IBM и другие разработали протокол безопасных электронных платежей (SEPP, *Secure Electronic Protocol*), фирмы Master Card и Visa International предложили технический стандарт по защите покупок по InterNet с платежом посредством карточек - *безопасный протокол транзакции* (SET, *Secure Electronic Transaction*).

8.3.СТАНДАРТЫ ЗАЩИТЫ ИНФОРМАЦИИ НА УРОВНЕ ОПЕРАЦИОННОЙ СИСТЕМЫ

Так как сетевое ПО в настоящее время обычно является частью ОС, к современным операционным системам (установленным на миллионах компьютеров, в том числе правительственных учреждений) правительства развитых стран предъявляют вполне определенные требования к уровню защиты информации, поддерживаемому самой ОС.

Первым правительством, установившем правила защиты для используемого в госучреждениях ПО (в том числе сетевого), явилось правительство США, установившее обязательные требования (такие, как защита ресурсов пользователя от несанкционированного доступа со стороны других пользователей, возможность установления квот на системные ресурсы и др.). Правительством США установлены уровни защиты от D (наименее строгий) до A (самый строгий); прохождение правительственной сертификации делает ОС конкурентоспособной в данной сфере.

Например, первоначально для известной ОС Windows'NT ставилась задача обеспечения уровня защиты C2 (определенного министерством обороны США как обеспечивающего 'селективное назначение прав доступа владельцев и, путем включения возможностей аудита, учет субъектов и инициируемых ими действий'); в будущих версиях возможно ужесточение уровня защиты данной ОС [5].

ЗАКЛЮЧЕНИЕ

Стартовав практически одновременно с созданием ПО общего назначения, программное обеспечение компьютерных сетей в настоящее время стало

существенной частью операционных систем, обеспечивая серьезное повышение функциональных возможностей ЭВМ путем объединения их ресурсов. Этот процесс имеет взрывной характер и привел к почти фантастическим последствиям (появлению сети InterNet).

Программное обеспечение компьютерных сетей, бывшее еще недавно весьма сложным и дорогим в создании и эксплуатации, сейчас вышло на уровень тривиального и является штатным компонентом операционных систем.

Следует ожидать разработки новых удобных специализированных протоколов высокого уровня, программных комплексов анализа и оптимизации сетей, перспективных приложений для InterNet и INTRANET, систем защиты информации в сетях.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Нанс Б. Компьютерные сети. Пер. с англ. -М.: БИНОМ, 1996. -395 с.
2. Сетевые средства Windows'NT. Пер с англ. -СПб.: BHV-Санкт-Петербург, 1996. -496 с.
3. Чаппел Л.,Хейкс Д. Анализ локальных сетей NetWare (фирменное руководство Novell). -М.: ЛОРИ, 1995. -596 с.
4. Нессер Д. Оптимизация и поиск неисправностей в сетях. -Киев.: Диалектика, 1996.
5. Кастер Х. Основы Windows'NT и NTFS. Пер. с англ. -М.: Русская редакция ТОО Channel Trading Ltd., 1996. -440 с.
6. Чепмен Д. и др. Разработка InterNet-приложений в DELPHI 2. -Киев.: DiaSoft, 1997. -640 с.
7. Елманова Н.З. Borland C++Builder (архитектура клиент/сервер, многозвенные системы, InterNet-приложения). -М.: Диалог-МИФИ, 1998. -240 с.
8. Зайцев С.С. Описание и реализация протоколов сетей ЭВМ. -М.: Наука, 1989. -272 с.
9. Клейнрок Л. Вычислительные системы с очередями. Пер. с англ. -М.: Мир, 1979. -600 с.
10. Фролов А.В., Фролов Г.В. Локальные сети персональных компьютеров (монтаж сети, установка программного обеспечения). Библиотека системного программиста, т.7. М.: Диалог-МИФИ, 1994. -169 с.
11. Фролов А.В., Фролов Г.В. Локальные сети персональных компьютеров (использование протоколов IPX, SPX, NetBIOS). Библиотека системного программиста, т.8. -М.: Диалог-МИФИ, 1995. -160 с.
12. Фролов А.В., Фролов Г.В. Локальные сети персональных компьютеров (работа с сервером Novell NetWare). Библиотека системного программиста, т.9. -М.: Диалог-МИФИ, 1993. -168 с.
13. Фролов А.В., Фролов Г.В.. Глобальные сети компьютеров (практическое введение в InterNet, E-Mail, FTP, WWW и HTML, программирование для

Windows Socket). Библиотека системного программиста, т.23. -М.: Диалог-МИФИ, 1996. -288 с.
14.Фролов А.В., Фролов Г.В. Программирование для Windows NT. Библиотека системного программиста, т.26/27. -М.: Диалог-МИФИ, 1996/1997. -272/271 с.
15.Фролов А.В., Фролов Г.В. Сервер WEB своими руками (язык HTML, приложения CGI и ISAPI, установка серверов WEB для Windows). Библиотека системного программиста, т.29. -М.: Диалог-МИФИ, 1998. -288 с.
16.Фролов А.В., Фролов Г.В. Microsoft Visual J++ (создание приложений и аплетов на языке Java, части 1 и 2). Библиотека системного программиста, т. 30/32. -М.: Диалог-МИФИ, 1997. -288/288 с.
17.Фролов А.В., Фролов Г.В. Разработка приложений для InterNet (Microsoft Visual C++ и MFC, среда Windows'95 и Windows'NT). Библиотека системного программиста, т.31. -М.: Диалог-МИФИ, 1997. -286 с.
18.Фролов А.В., Фролов Г.В. JavaScript (сценарии JavaScript в активных страницах WEB). Библиотека системного программиста, т.34. -М.: Диалог-МИФИ, 1998. -284 с.
19.Морис Б. HTML в действии (секреты сценариев, передовые технологии, средства ActiveX, Java). Пер. с англ. -СПб.: Питер, 1997. -256 с.
20.Холзнер С. PERL: специальный справочник. Пер. с англ. -СПб.: Питер, 2000. -496 с.
21.Родли Д. Создание JAVA-апплетов. Пер. с англ. -Киев.: DiaSoft, 1996. -384 с.
22.Авраамова О.Д. Язык VRML (практическое руководство). -М.: Диалог-МИФИ, 2000. -288 с.
23.Мещеряков Е.В., Хомоненко А.Д. Публикация баз данных в Интернете. - СПб.: БХВ-Петербург, 2001. -560 с.
24.Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002. -608 с.