
*Томский межвузовский центр дистанционного образования*

**И.В. Бойченко**

# ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СЕТЕЙ ЭВМ

**Учебное пособие**

**ТОМСК – 2005**

Корректор: Миллер С.В.

**Бойченко И.В.**
Программное обеспечение сетей ЭВМ: Учебное пособие. — Томск: Томский межвузовский центр дистанционного образования, 2005. — 294 с.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ ..... 5
1 ПРИМЕНЕНИЕ СОВРЕМЕННЫХ СЕТЕЙ ЭВМ ..... 6
2 АРХИТЕКТУРА И СТАНДАРТЫ СЕТЕЙ ЭВМ ..... 11
2.1 Архитектура «клиент-сервер» и сетевые операционные системы ..... 11
2.2 Обзор сетевых операционных систем ..... 19
2.3 Стандарты сетей ЭВМ ..... 30
2.4 Модели сетевого взаимодействия ..... 40
2.4.1 Модель OSI (BOC) ..... 40
2.4.2 Модель DoD. Сравнение с моделью OSI ..... 43
3 ОПИСАНИЕ СЕРВИСА В МОДЕЛИ OSI ..... 46
3.1 Общие сведения ..... 46
3.2 Функции уровней ..... 49
3.3 Представление сервиса в модели BOC ..... 59
3.4 Правила описания сервиса ..... 62
3.5 Взаимосвязь системы в режиме «без соединения» ..... 68
3.6 Физический уровень ..... 70
3.7 Канальный уровень ..... 73
3.7.1 Канальный сервис с соединением ..... 74
3.7.2 Параметры качества сервиса ..... 84
3.7.3 Канальный сервис без соединения ..... 87
3.8 Сетевой уровень ..... 89
3.8.1 Сетевой сервис с соединением ..... 92
3.8.2 Параметры качества сервиса ..... 96
3.8.3 Сетевой сервис без соединения ..... 98
3.9 Сервис транспортного уровня ..... 99
3.9.1 Транспортный сервис с соединением ..... 105
3.9.2 Качество транспортного сервиса ..... 107
3.9.3 Транспортный сервис без соединения ..... 108
3.10 Сервис сеансового уровня ..... 109
3.11 Сервис представительного уровня ..... 122
3.11.1 Спецификация синтаксиса для верхних уровней ..... 126
3.12 Общий прикладной сервис ..... 143
3.12.1 Сервис обеспечения неделимости распределенных действий ..... 146



4 СТЕКИ ПРОТОКОЛОВ И ПРОТОКОЛЫ СЕТЕЙ ЭВМ ..... 151
4.1 Обзор существующих стеков протоколов ..... 151
4.2 Стек TCP/IP ..... 157
4.2.1 Адресация в TCP/IP и IPX/SPX ..... 160
4.2.2 Система доменных имен DNS ..... 166
4.2.3 Интернет протокол-IP ..... 180
4.2.4 Протокол контроля сообщений — ICMP ..... 187
4.2.5 Протокол датаграмм пользователя — UDP ..... 191
4.2.6 Протокол контроля передачи — TCP ..... 192
4.3 Программирование сокетов Беркли ..... 204
4.3.1 Примитивы сокетов Беркли ..... 204
4.3.2 Сервер ..... 205
4.3.3 Клиент ..... 207
5 ПРИКЛАДНОЙ УРОВЕНЬ СТЕКА TCP/IP ..... 209
5.1 Электронная почта ..... 209
5.2 Всемирная паутина (WWW) ..... 242
5.2.1 Архитектура WWW ..... 244
5.2.2 Статические веб-документы ..... 262
5.2.3 Динамические веб-документы ..... 282
СПИСОК ЛИТЕРАТУРЫ ..... 294

## ВВЕДЕНИЕ

Курс «Программное обеспечение сетей ЭВМ» входит в региональную компоненту учебного образовательного стандарта подготовки специалистов по специальностям 220400 «Программное обеспечение вычислительной техники и автоматизированных систем», и 351400 «Прикладная информатика в экономике».

Предметом данной дисциплины является изучение технологий передачи информации, основанное на представлении эталонной модели взаимосвязи открытых систем. Учебное пособие состоит из пяти глав. В трех первых главах рассматриваются общие вопросы организации программного обеспечения сетей ЭВМ, принципы межуровнего взаимодействия, описывается необходимый для этого сервис. Четвертая и пятая главы посвящены описанию и сравнению существующих стеков сетевых протоколов. Подробно описан стек интернет-протоколов TCP/IP, затронуты вопросы программирования сокетов (sockets). В пятой главе рассматриваются протоколы прикладного уровня TCP/IP.

Предполагается, что учащийся уже имеет знания и навыки, полученные в ходе изучения дисциплин «Вычислительные машины, сети и системы телекоммуникаций» (либо «Архитектура вычислительных систем и сетей»), «Операционные системы», «Программирование».

Данное учебное пособие является результатом коллективного труда. При подготовке учебного пособия большую помощь оказали студенты группы 439 Факультета систем управления ТУСУР, и, в особенности, Елена Ефремова и Ирина Гузовяко.

Отдельное большое спасибо Бойченко Юлии Анатольевне за помощь в подготовке учебника к изданию.



## 1 ПРИМЕНЕНИЕ СОВРЕМЕННЫХ СЕТЕЙ ЭВМ

Вычислительные сети (ВС) — системы для передачи и совместной обработки данных с помощью независимых ЭВМ. ВС являются системами со слабыми программными и аппаратными связями. Основными элементами сети являются стандартные компьютеры, не имеющие ни общих блоков памяти, ни периферийных устройств. Тогда как внутри ЭВМ программные и аппаратные связи сильны, что необходимо для обеспечения работоспособности. Взаимодействие между компьютерами сети происходит за счет передачи сообщений через сетевые адаптеры и каналы связи. С помощью этих сообщений ЭВМ запрашивают и поставляют услуги или сервисы. Запрашивающая ЭВМ называется *клиентом*, ЭВМ, поставляющая услуги, называется *сервером*. В общем случае, в зависимости от вида сервиса, одна и та же ЭВМ может выступать как в качестве клиента, так и в качестве сервера.

В ВС могут существовать следующие виды сервисов или услуг:

- разделение, совместное использование (sharing) дорогостоящего периферийного оборудования (принтеры, плоттеры, модемы);
- доступ к различного вида информации (файловый сервис, сервис БД, ресурсы WWW);
- коммуникации (сервис передачи и обмена сообщениями, электронная почта). Для многих предприятий именно этот сетевой сервис является определяющим фактором для использования сети;
- выполнение параллельных вычислений, удаленных вычислений (remote computing), сервис приложений, когда задачи, требующие интенсивных вычислений, выполняются на более мощной ЭВМ, а другая ЭВМ использует эти результаты вычислений.

Это основные разновидности различных сетевых служб. Современное программное обеспечение часто является комбинацией вышеприведенных возможностей. Особенность ВС как системы — это распределённость. Следствием этого свойства является более высокая отказоустойчивость по сравнению с централизо-

ванными системами, возможность организации параллельных вычислений, возможность более удобного размещения узлов обработки данных. Как мы уже отметили, взаимодействие между компьютерами сети происходит за счет передачи сообщений через сетевые адаптеры и каналы связи. Особенности и ограничения, связанные с передачей данных, часто являются принципиальными и влияют на концепции построения ВС. В зависимости от территориальной протяженности ВС делят на локальные (ЛВС) и глобальные (ГВС).

Основные отличия между ЛВС и ГВС:

- скорость передачи сообщений между конечными узлами;
- диаметр сети или протяженность каналов связи;
- конфигурация связей;
- количество узлов сети.

Исторически глобальные сети возникли раньше, что связано с эволюцией ЭВМ. Первые ЭВМ — большие системы, обычно не более одной на организацию или научный центр. Со временем возникла необходимость обмена данными между научными центрами или госучреждениями, то есть передача информации на большие расстояния. Эти причины привели к созданию глобальных сетей, охватывающих любые пространства. Линии связи дороги, а пользователей много. Следовательно, нужно уменьшить количество передаваемой вспомогательной информации. В локальных сетях эта проблема остро не стоит. Стоимость передачи единицы информации здесь гораздо меньше, так как каналы связи принадлежат только одному предприятию, расстояния гораздо меньше и технологии передачи дешевле.

Соответственно и скорость гораздо выше между конечными узлами в ЛВС. В частности, это относится к проблеме затухания сигнала. Если расстояние большое, то сигнал следует создавать в виде гармоники определенной частоты, попадающей в полосу пропускания данного проводника (медь и подобные). Кодирование и декодирование сигнала усложняется. А в локальных сетях можно передавать импульсы прямоугольной формы, так как расстояние между узлами — сотни метров. На большое расстояние такой сигнал передать невозможно. Прямоугольные сигналы легче декодировать, и скорость передачи может быть выше.



Что касается конфигурации связей, то в ГВС конфигурацию можно назвать случайной, то есть между двумя узлами может быть множество путей. В ЛВС обычно используются стандартные технологии, достаточно жестко фиксирующие вид и конфигурацию связей между узлами. Наиболее часто встречаются следующие конфигурации:

- звезда;
- шина;
- кольцо;
- древовидная;
- комбинированные.

Отсюда можно сделать вывод, что задача маршрутизации в глобальных сетях сложнее, чем в локальных.

И последнее, количество узлов в глобальной сети неограниченно, в то время как в локальных сетях количество узлов ограничивается применяемой стандартной технологией во избежание такой проблемы, как перенаселенность сети (congestion error).

Из проведенного сравнения видно, что в глобальных сетях многие вопросы решать сложнее, что отражается на качестве и разнообразии сетевых служб. Так, например, в глобальных сетях достаточно сложно организовать процесс обработки данных в реальном режиме времени. Решение такой задачи, как просмотр всех имеющихся сетевых ресурсов, проблематично, а в локальной сети тривиально. После соединения с удаленным ресурсом в ЛВС пользователь может работать с помощью известного набора команд и прикладного ПО, тогда как работа в ГВС требует определенного набора специфических знаний. Особенно это отличие было заметно на первоначальном этапе развития ВС.

Несмотря на различия, глобальные и локальные сети являются разновидностями систем одного класса — вычислительных сетей. Поэтому при создании ЛВС и ГВС приходится решать одни и те же задачи, но, возможно, в разных масштабах и разными средствами. В определенной степени ЛВС являются подмножеством ГВС, так как возникли позднее и переняли от ГВС ряд уже созданных технологий. В настоящее время наблю-

дается тенденция к сближению технологий ГВС и ЛВС. Предпосылки этого сближения изложены ниже.

Во-первых, увеличивается пропускная способность каналов связи, используемых глобальными сетями. Вследствие этого появляется возможность организации работы в глобольных сетях с приближением к реальному режиму времени. Технологически это основывается на средствах оптической передачи данных. Цена оптоволоконных проводников сравнилась с электрическими проводниками высших категорий (витая пара 6 категории, максимальная пропускная способность ~1 Гб/с при длине 100 м, в то время как эта же характеристика оптоволоконных проводников ~10 Гб/с на 2–3 км). Однако препятствия заключаются в дороговизне оборудования и монтажа, со временем и эта проблема будет снята.

Во-вторых, в локальных сетях возникают проблемы масштабирования из-за ограничений на количество узлов в сети. С возникновением корпоративных сетей появилась необходимость решения, позволяющего обойти эти ограничения. В глобальных сетях подобная задача решалась изначально. Многие из этих решений переносятся в ЛВС. Одним из таких примеров может служить перенос стека протоколов TCP/IP в локальные сети. Этот протокол обеспечивает широкие возможности по масштабированию сетей, однако, требует более сложного администрирования.

На заре современных локальных сетей в 80-е годы рабочие станции не были настолько мощны, чтобы программно поддерживать функциональность стека TCP/IP. Поэтому использовались более простые и эффективные протоколы (IPX/SPX). Современные ПК превосходят своих предшественников по производительности в десятки и сотни раз, и это ограничение не играет роли. Наряду с технологиями низкого уровня переносятся и технологии высокого уровня. Возникло такое понятие, как сеть *интранет,* то есть локальные или корпоративные, в своей основе, сети, но использующие такие технологии, как WWW, электронную почту, FTP для решения внутренних задач. Выгоды поддержки технологий глобальных сетей очевидны. Эта отрасль бурно развивается, появляются новые технологии, причем на открытой основе, то есть распространяются фактически бесплатно. Этот конгломерат технологий способен решать самые разные задачи. Создав в своих программных продуктах возможности работы с технологиями ГВС, производители обеспечивают успех за счет совместимости с другими продуктами и возможности дальнейшего развития.

# 2 АРХИТЕКТУРА И СТАНДАРТЫ СЕТЕЙ ЭВМ

## 2.1 Архитектура «клиент-сервер» и сетевые операционные системы

Сетевая операционная система (ОС) составляет основу любой вычислительной сети. Каждый компьютер в сети в значительной степени автономен, поэтому под сетевой операционной системой в широком смысле понимается совокупность операционных систем отдельных компьютеров, взаимодействующих с целью обмена сообщениями и разделения ресурсов по единым правилам — протоколам. В узком смысле сетевая ОС — это операционная система отдельного компьютера, обеспечивающая ему возможность работать в сети.

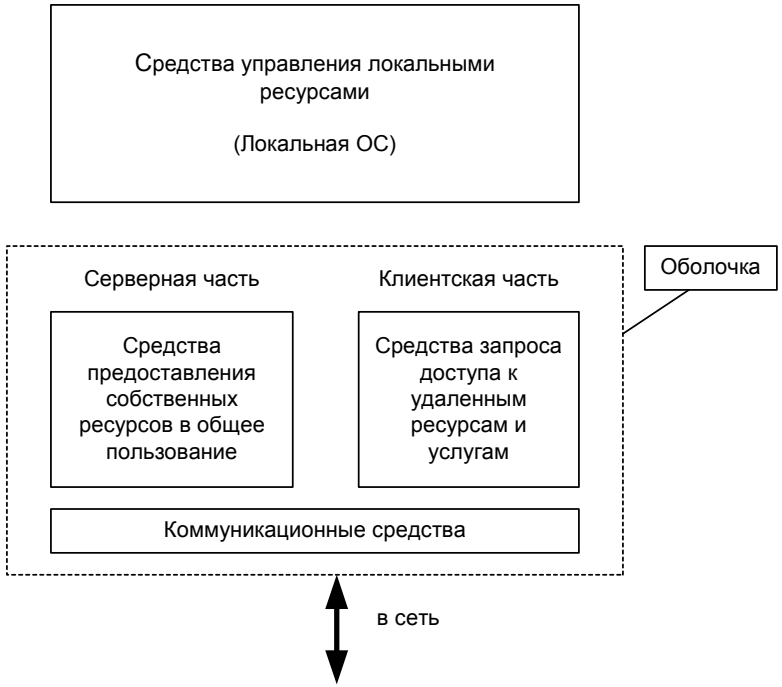



Рисунок 2.1 — Структура сетевой ОС



В сетевой операционной системе отдельной машины можно выделить несколько частей (рис. 2.1):

Средства управления локальными ресурсами компьютера: функции распределения оперативной памяти между процессами, планирования и диспетчеризации процессов, управления процессорами в мультипроцессорных машинах, управления периферийными устройствами и другие функции управления ресурсами локальных ОС.

Средства предоставления собственных ресурсов и услуг в общее пользование — серверная часть ОС (сервер). Эти средства обеспечивают, например, блокировку файлов и записей, что необходимо для их совместного использования; ведение справочников имен сетевых ресурсов; обработку запросов удаленного доступа к собственной файловой системе и базе данных; управление очередями запросов удаленных пользователей к своим периферийным устройствам.

Средства запроса доступа к удаленным ресурсам и услугам и их использования — клиентская часть ОС (редиректор). Эта часть выполняет распознавание и перенаправление в сеть запросов к удаленным ресурсам от приложений и пользователей, при этом запрос поступает от приложения в локальной форме, а передается в сеть в другой форме, соответствующей требованиям сервера. Клиентская часть также осуществляет прием ответов от серверов и преобразование их в локальный формат, так что для приложения выполнение локальных и удаленных запросов неразличимо.

Коммуникационные средства ОС. С их помощью происходит обмен сообщениями в сети. Эта часть обеспечивает адресацию и буферизацию сообщений, выбор маршрута передачи сообщения по сети, надежность передачи и т.п., то есть является средством транспортировки сообщений.

В зависимости от функций, возлагаемых на конкретный компьютер, в его операционной системе может отсутствовать либо клиентская, либо серверная части. Рассмотрим взаимодействие сетевых компонентов (рис. 2.2). «Компьютер 1» выполняет роль «чистого» клиента, а «Компьютер 2» — роль «чистого» сервера, соответственно на первой машине отсутствует серверная часть, а на второй — клиентская. На рисунке отдельно пока-

зан компонент клиентской части — редиректор. Именно редиректор перехватывает все запросы, поступающие от приложений, и анализирует их. Если выдан запрос к ресурсу данного компьютера, то он переадресовывается соответствующей подсистеме локальной ОС, если же это запрос к удаленному ресурсу, то он переправляется в сеть. При этом клиентская часть преобразует запрос из локальной формы в сетевой формат и передает его транспортной подсистеме, которая отвечает за доставку сообщений указанному серверу. Серверная часть операционной системы «Компьютера 2» принимает запрос, преобразует его и передает для выполнения своей локальной ОС. После того, как результат получен, сервер обращается к транспортной подсистеме и направляет ответ клиенту, выдавшему запрос. Клиентская часть преобразует результат в соответствующий формат и адресует его тому приложению, которое выдало запрос.

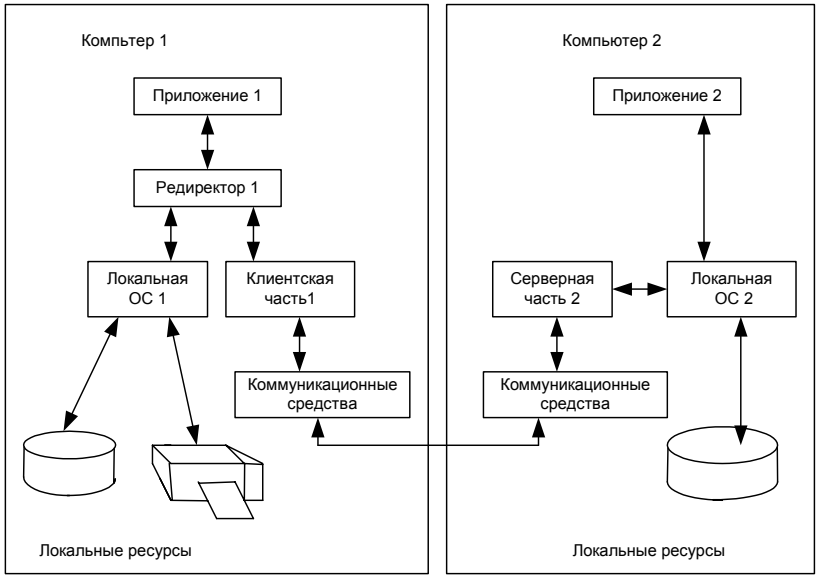



Рисунок 2.2 — Взаимодействие компонентов операционной системы при взаимодействии компьютеров



На практике сложилось несколько подходов к построению сетевых операционных систем (рис. 2.3).

Первые сетевые ОС представляли собой совокупность существующей локальной ОС и надстроенной над ней *сетевой оболочки*. При этом в локальную ОС встраивался минимум сетевых функций, необходимых для работы сетевой оболочки, которая выполняла основные сетевые функции. Примером такого подхода является использование на каждой машине сети операционной системы MS DOS (у которой, начиная с ее третьей версии, появились такие встроенные функции, как блокировка файлов и записей, необходимые для совместного доступа к файлам). Принцип построения сетевых ОС в виде сетевой оболочки над локальной ОС используется и в современных ОС, таких, например, как LANtastic или Personal Ware.

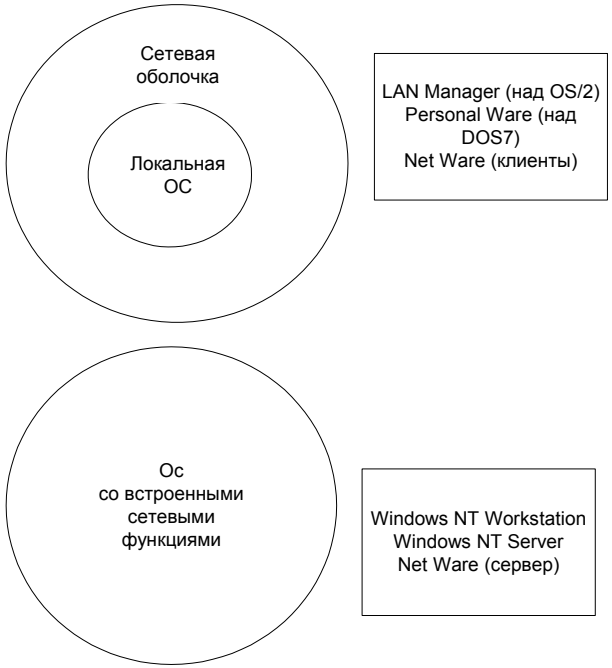



Рисунок 2.3 — Варианты построения сетевых ОС

Однако более эффективным представляется путь разработки операционных систем, изначально предназначенных для работы в сети. Сетевые функции у ОС такого типа глубоко *встроены* в основные модули системы, что обеспечивает их логическую стройность, простоту эксплуатации и модификации, а также высокую производительность. Примером такой ОС является система Windows NT фирмы Microsoft, которая за счет встроенности сетевых средств обеспечивает более высокие показатели производительности и защищенности информации по сравнению с сетевой ОС LAN Manager той же фирмы (совместная разработка с IBM), являющейся надстройкой над локальной операционной системой OS/2.

*Одноранговые сетевые ОС и ОС с выделенными серверами*

В зависимости от того, как распределены функции между компьютерами сети, сетевые операционные системы, а, следовательно, и сети делятся на два класса: одноранговые и двухранговые (рис. 2.4). Последние чаще называют сетями с выделенными серверами.

Если компьютер предоставляет свои ресурсы другим пользователям сети, то он играет роль сервера. При этом компьютер, обращающийся к ресурсам другой машины, является клиентом. Как уже было сказано, компьютер, работающий в сети, может выполнять функции либо клиента, либо сервера, либо совмещать обе эти функции.

Если выполнение каких-либо серверных функций является основным назначением компьютера (например, предоставление файлов в общее пользование всем остальным пользователям сети, организация совместного использования факса или предоставление всем пользователям сети возможности запуска на данном компьютере своих приложений), то такой компьютер называется *выделенным сервером*. В зависимости от того, какой ресурс сервера является разделяемым, он называется файл-сервером, факс-сервером, принт-сервером, сервером приложений и т.д.

Очевидно, что на выделенных серверах желательно устанавливать ОС, специально оптимизированные для выполнения тех или иных серверных функций. Поэтому в сетях с выделенными серверами чаще всего используются сетевые операционные системы, в состав которых входит нескольких вариантов ОС, отличающихся возможностями серверных частей.



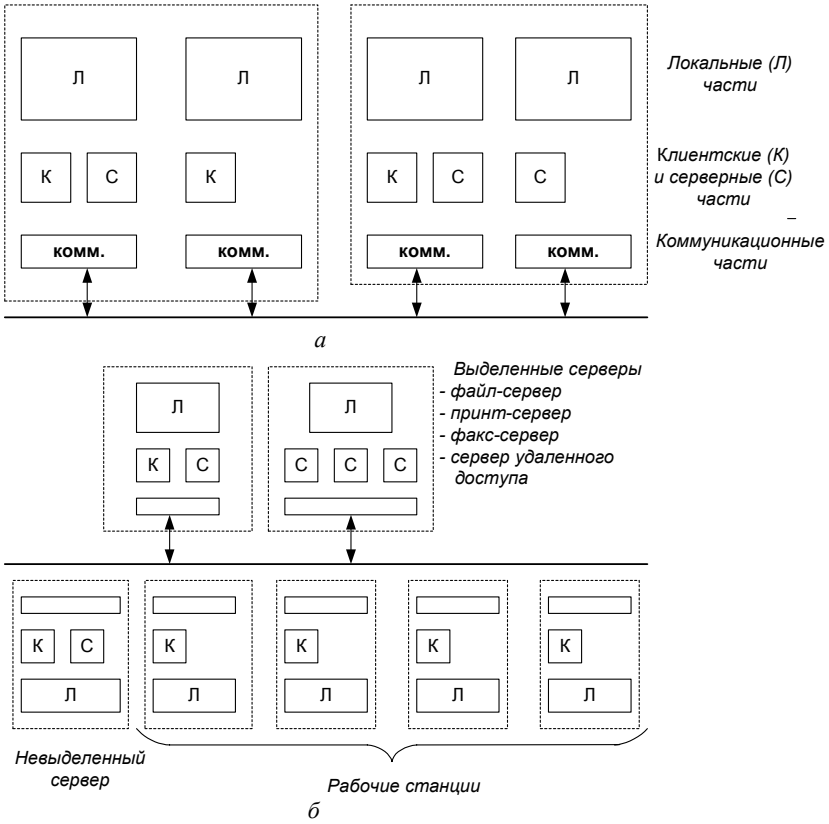



Рисунок 2.4
*а* — одноранговая сеть, *б* — двухранговая сеть

Например, сетевая ОС Novell NetWare имеет серверный вариант, оптимизированный для работы в качестве файл-сервера, а также варианты оболочек для рабочих станций с различными локальными ОС, причем эти оболочки выполняют исключительно функции клиента. Другим примером ОС, ориентированной на построение сети с выделенным сервером, является операционная система Windows NT. В отличие от NetWare, оба варианта данной сетевой ОС — Windows NT Server (для выделенного сервера) и Windows NT Workstation (для рабочей станции) — могут поддерживать функции и клиента и сервера.

Но серверный вариант Windows NT имеет больше возможностей для предоставления ресурсов своего компьютера другим пользователям сети, так как может выполнять более широкий набор функций, поддерживает большее количество одновременных соединений с клиентами, реализует централизованное управление сетью, имеет более развитые средства защиты.

Выделенный сервер не принято использовать в качестве компьютера для выполнения текущих задач, не связанных с его основным назначением, так как это может уменьшить производительность его работы как сервера. В связи с такими соображениями в ОС Novell NetWare на серверной части возможность выполнения обычных прикладных программ вообще не предусмотрена, то есть сервер не содержит клиентской части, а на рабочих станциях отсутствуют серверные компоненты. Однако в других сетевых ОС функционирование на выделенном сервере клиентской части вполне возможно. Например, под управлением Windows NT Server могут запускаться обычные программы локального пользователя, которые могут потребовать выполнения клиентских функций ОС при появлении запросов к ресурсам других компьютеров сети. При этом рабочие станции, на которых установлена ОС Windows NT Workstation, могут выполнять функции невыделенного сервера.

Важно понять, что, несмотря на то, что в сети с выделенным сервером все компьютеры в общем случае могут выполнять одновременно роли и сервера, и клиента, эта сеть функционально не симметрична. Аппаратно и программно в ней реализованы два типа компьютеров: одни в большей степени ориентированы на выполнение серверных функций и работают под управлением специализированных серверных ОС, а другие — в основном выполняют клиентские функции и работают под управлением соответствующего этому назначению варианта ОС. Функциональная несимметричность, как правило, вызывает и несимметричность аппаратуры — для выделенных серверов используются более мощные компьютеры с большими объемами оперативной и внешней памяти. Таким образом, функциональная несимметричность в сетях с выделенным сервером сопровождается несимметричностью операционных систем (специализация ОС) и аппаратной несимметричностью (специализация компьютеров).



В одноранговых сетях все компьютеры равны в правах доступа к ресурсам друг друга. Каждый пользователь может по своему желанию объявить какой-либо ресурс своего компьютера разделяемым, после чего другие пользователи могут его эксплуатировать. В таких сетях на всех компьютерах устанавливается одна и та же ОС, которая предоставляет всем компьютерам в сети *потенциально* равные возможности. Одноранговые сети могут быть построены, например, на базе ОС LANtastic, Personal Ware, Windows for Workgroup, Windows NT Workstation.

В одноранговых сетях также может возникнуть функциональная несимметричность: одни пользователи не желают разделять свои ресурсы с другими, и в таком случае их компьютеры выполняют роль клиента; за другими компьютерами администратор закрепил только функции по организации совместного использования ресурсов, а значит, они являются серверами. В третьем случае, когда локальный пользователь не возражает против использования его ресурсов и сам не исключает возможности обращения к другим компьютерам, ОС, устанавливаемая на его компьютере, должна включать и серверную, и клиентскую части. В отличие от сетей с выделенными серверами, в одноранговых сетях отсутствует специализация ОС в зависимости от преобладающей функциональной направленности — клиента или сервера. Все вариации реализуются средствами конфигурирования одного и того же варианта ОС.

Одноранговые сети проще в организации и эксплуатации, однако, они применяются в основном для объединения небольших групп пользователей, не предъявляющих больших требований к объемам хранимой информации, ее защищенности от несанкционированного доступа и к скорости доступа. При повышенных требованиях к этим характеристикам более подходящими являются двухранговые сети, где сервер лучше решает задачу обслуживания пользователей своими ресурсами, так как его аппаратура и сетевая операционная система специально спроектированы для этой цели.

## 2.2 Обзор сетевых операционных систем

Критериями для выбора ОС масштаба предприятия являются следующие характеристики:

- поддержка многосерверной сети;
- высокая эффективность файловых операций;
- возможность эффективной интеграции с другими ОС;
- наличие централизованной масштабируемой справочной службы;
- хорошие перспективы развития;
- эффективная работа удаленных пользователей;
- разнообразные сервисы: файл-сервис, принт-сервис, безопасность данных и отказоустойчивость, архивирование данных, служба обмена сообщениями, разнообразные базы данных и другие;
- разнообразные программно-аппаратные хост-платформы: IBM SNA, DEC NSA, UNIX;
- разнообразные транспортные протоколы: TCP/IP, IPX/SPX, NetBIOS, AppleTalk;
- поддержка многообразных операционных систем конечных пользователей: DOS, UNIX, OS/2, Mac;
- поддержка сетевого оборудования стандартов Ethernet, Token Ring, FDDI, ARCnet;
- наличие популярных прикладных интерфейсов и механизмов вызова удаленных процедур RPC;
- возможность взаимодействия с системой контроля и управления сетью, поддержка стандартов управления сетью SNMP.

ОС в наибольшей степени определяет облик всей вычислительной системы в целом. ОС выполняет две, по существу, мало связанные функции: обеспечение пользователю-программисту удобств посредством предоставления для него расширенной машины и повышение эффективности использования компьютера путем рационального управления его ресурсами.

Выполняя первую функцию, ОС ограждает программистов, например, от аппаратуры дискового накопителя и предоставляет простой файловый интерфейс. ОС берет на себя все малоприятные дела, связанные с обработкой прерываний, управлением



таймерами и оперативной памятью, а также другие низкоуровневые проблемы. В каждом случае та абстрактная, воображаемая машина, с которой, благодаря операционной системе, теперь может иметь дело пользователь, гораздо проще и удобнее в обращении, чем реальная аппаратура, лежащая в основе этой абстрактной машины.

Второй функцией ОС является распределение процессоров, памяти, устройств и данных между процессами, конкурирующими за эти ресурсы. ОС должна управлять всеми ресурсами вычислительной машины таким образом, чтобы обеспечить максимальную эффективность ее функционирования. Критерием эффективности может быть, например, пропускная способность или реактивность системы. Управление ресурсами включает решение двух общих, не зависящих от типа ресурса задач:

- планирование ресурса — то есть определение кому, когда, а для делимых ресурсов и в каком количестве, необходимо выделить данный ресурс;
- отслеживание состояния ресурса — то есть поддержание оперативной информации о том, занят или не занят ресурс, а для делимых ресурсов — какое количество ресурса уже распределено, а какое свободно.

Для решения этих общих задач управления ресурсами разные ОС используют различные алгоритмы, что, в конечном счете, и определяет их облик в целом, включая характеристики производительности, область применения и даже пользовательский интерфейс. Так, например, алгоритм управления процессором в значительной степени определяет, является ли ОС системой разделения времени, системой пакетной обработки или системой реального времени. Рассмотрим примеры существующих популярных сетевых операционных систем.

Этот обзор не претендует на стопроцентный охват этой предметной области. В наши задачи входит дать описание типичных представителей операционных систем, использующихся в современных вычислительных сетях.

*UNIX и UNIX-подобные системы*

Широкое распространение UNIX породило проблему несовместимости его многочисленных версий. Очевидно, что для

пользователя весьма неприятен тот факт, что пакет, купленный для одной версии UNIX, отказывается работать на другой версии UNIX. Периодически делались и делаются попытки стандартизации UNIX, но они пока имели ограниченный успех. Процесс сближения различных версий UNIX и их расхождения носит циклический характер. Перед лицом новой угрозы со стороны какой-либо другой операционной системы различные производители UNIX-версий сближают свои продукты, но затем конкурентная борьба вынуждает их делать оригинальные улучшения и версии снова расходятся. В этом процессе есть и положительная сторона — появление новых идей и средств, улучшающих как UNIX, так и многие другие операционные системы, перенявшие у него за долгие годы его существования много полезного.

Рассмотрим упрощенную схему развития UNIX систем, которая учитывает преемственность различных версий и влияние на них принимаемых (рис. 2.5). Наибольшее распространение получили две весьма несовместимые линии версий UNIX: линия AT&T — UNIX System V, и линия университета Berkeley-BSD. Многие фирмы на основе этих версий разработали и поддерживают свои версии UNIX: SunOS и Solaris фирмы Sun Microsystems, UX фирмы Hewlett-Packard, XENIX фирмы Microsoft, AIX фирмы IBM, UnixWare фирмы Novell (проданный теперь компании SCO), и список этот можно еще долго продолжать.

Наибольшее влияние на унификацию версий UNIX оказали такие стандарты как SVID фирмы AT&T, POSIX, созданный под эгидой IEEE, и XPG4 консорциума X/Open. В этих стандартах сформулированы требования к интерфейсу между приложениями и ОС, что дает возможность приложениям успешно работать под управлением различных версий UNIX. Независимо от версии, общими для UNIX чертами являются:

- многопользовательский режим со средствами защиты данных от несанкционированного доступа,
- реализация мультипрограммной обработки в режиме разделения времени, основанная на использовании алгоритмов, вытесняющей многозадачности (preemptive multitasking),



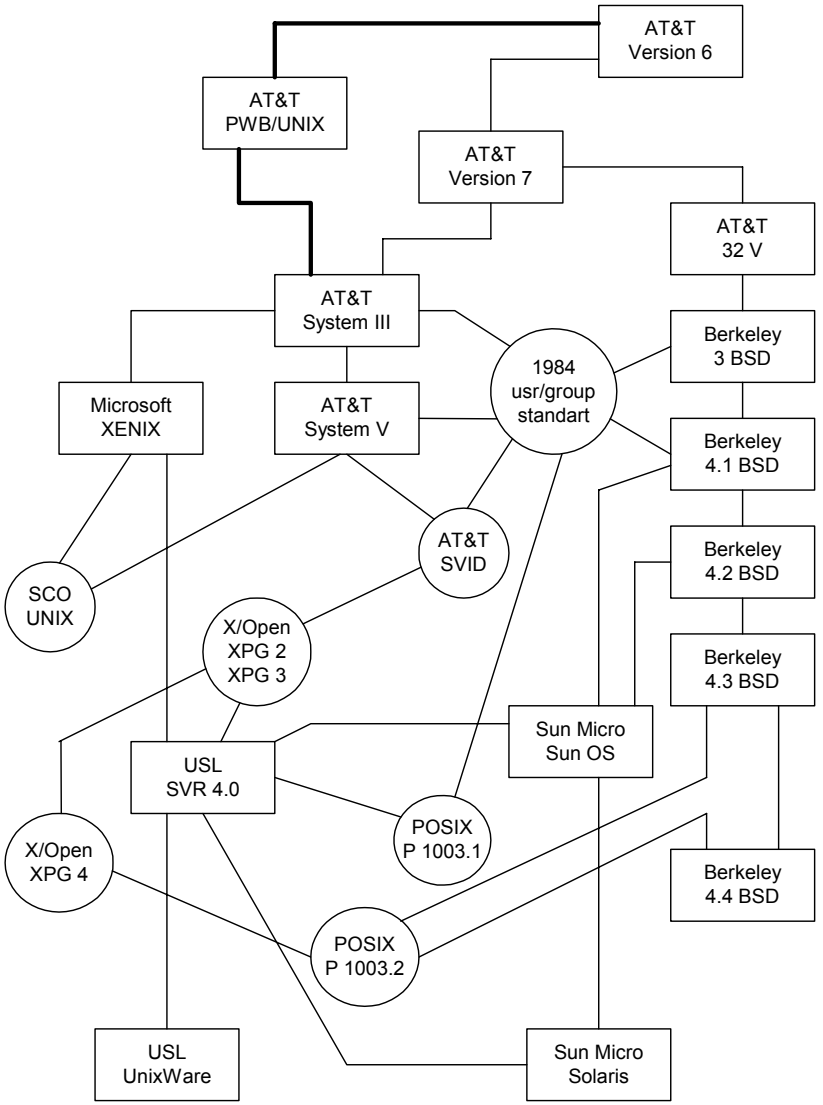



Рисунок 2.5 — История развития UNIX

- использование механизмов виртуальной памяти и свопинга для повышения уровня мультипрограммирования,
- унификация операций ввода-вывода на основе расширенного использования понятия «файл»,
- иерархическая файловая система, образующая единое дерево каталогов независимо от количества физических устройств, используемых для размещения файлов,
- переносимость системы за счет написания ее основной части на языке С,
- разнообразные средства взаимодействия процессов, в том числе и через сеть,
- кэширование диска для уменьшения среднего времени доступа к файлам.

Версия UNIX SVR4.2 была создана фирмой UNIX System Laboratories (USL) в 1992 году как развитие версии UNIX System V Release 4. Для совместимости этой версии с наиболее популярными в секторе локальных сетей операционными системами Novell NetWare было создано совместное предприятие USL и Novell Univel, которое разработало и выпустило на рынок операционную систему UnixWare.

Все большую популярность в настоящее время завоевывает Linux — система, ядро которой разработано в начале 90-х годов. На сегодняшний день существует несколько десятков разновидностей и клонов Linux: Red Hat, SUSE, ASP, Mandrake, Ximian, Debian, ALT и др.

*Novell NetWare*

Novell — это крупнейшая фирма, которой принадлежит, согласно различным источникам, от 65% до 75% рынка сетевых операционных систем для локальных вычислительных сетей. Наибольшую известность фирма Novell приобрела благодаря своим сетевым операционным системам семейства NetWare. Эти системы реализованы как системы с выделенными серверами.

Основные усилия Novell были затрачены на создание высокоэффективной серверной части сетевой ОС, которая за счет специализации на выполнении функций файл-сервера обеспечивала бы максимально возможную для данного класса компьютеров скорость удаленного доступа к файлам и повышенную безо-



пасность данных. Для серверной части своих ОС Novell разработала специализированную операционную систему, оптимизированную на файловые операции и использующую все возможности, предоставляемые процессорами Intel x386 и выше. За высокую производительность пользователи сетей Novell NetWare расплачиваются стоимостью — выделенный файл-сервер не может использоваться в качестве рабочей станции, а его специализированная ОС имеет весьма специфический API (Application Program Interface — Интерфейс прикладных программ), что требует от разработчиков серверных модулей особых знаний, и специального опыта.

Для рабочих станций Novell выпускала две собственные ОС со встроенными сетевыми функциями: Novell DOS 7 с входящей в нее сетевой одноранговой компонентой Personal Ware, а также ОС UnixWare, являющейся реализацией UNIX System V Release 4.2 со встроенными возможности работы в сетях NetWare. В последствии ОС UnixWare была продана компании Santa Cruz Operations. Для популярных ОС персональных компьютеров других производителей Novell выпускает сетевые оболочки с клиентскими функциями (обычно говорят просто — «клиенты») по отношению к серверу NetWare.

Первоначально операционная система NetWare была разработана фирмой Novell для сети Novell S-Net, имеющей звездообразную топологию и патентованный сервер с микропроцессором Motorola MC68000. Когда фирма IBM выпустила персональные компьютеры типа PC XT, Novell решила, что NetWare может быть легко перенесена на архитектуру микропроцессоров семейства Intel 8088 и 8086, и тогда она сможет поддерживать практически все имеющиеся на рынке сети персональных компьютеров. Это решение принесло Novell очень широкую популярность. В середине 80-х годов двадцатого века вопрос о том, какой сервер устанавливать в локальных сетях персональных компьютеров имел однозначный ответ: только Novell Netware. История развития NetWare отражена в таблице 2.1.

NetWare — специализированная операционная система, оптимизированная для работы в качестве файлового сервера и принт-сервера. Отсутствуют API основных операционных сред, используемых для разработки приложений, — UNIX, Windows,

OS/2. Это значит, что никакое программное обеспечение запустить в среде NetWare нельзя, кроме того, которое создано специально для этой ОС.

Таблица 2.1 — История развития Novell Netware

| Название | Версия | Год выпуска | Особенности |
|---|---|---|---|
| NetWare | 1.0 | 1983 | переход на i8088, i8086 |
| Advanced NetWare | 1.0 | 1985 | |
| | 1.2 | 1985 | поддержка защищенного режима i80286 |
| | 2.0 | 1986 | полная поддержка защищенного режима i80286; высокая производительность; возможность подключения к одному серверу до четырех сетей с различной топологией, таких как Ethernet, ArcNet и Token Ring |
| NetWare SFT (System Fault Tolerance) отказоустойчивая | 2.10 | 1987 | учет используемых ресурсов и защита от несанкционированного доступа, разработчики впервые получили возможность создавать многопользовательские прикладные программы, которые могут выполняться на сервере в качестве дополнительных процессов сетевой операционной системы и использовать ее функциональные возможности |
| NetWare | 2.15 | 1988 | пользователи Macintosh получили возможность подключать свои компьютеры в качестве клиентов серверов NetWare, получая доступ к ресурсам сети и осуществляя прозрачный поиск и хранение информации на сервере, также обеспечивается устойчивость к сбоям и защита от несанкционированного доступа |
| NetWare 386 | 3.0 | 1989 | первая версия полностью 32-разрядной сетевой операционной системы для серверов на базе ПК; значительно более высокая производительность, усовершенствованная система защиты от несанкционированного доступа, отвечала самым передовым требованиям к среде функционирования распределенных прикладных программ |



Продолжение табл. 2.1

| Название | Версия | Год выпуска | Особенности |
|---|---|---|---|
| | 3.1 | 1990 | повышена производительность, улучшены инструментальные средства для независимых разработчиков. |
| NetWare | 2.2 | 1991 | замена операционных систем для процессоров 80286 (SFT, Advanced и ELS NetWare) на более мощную и удобную систему, функционально превосходящую предыдущие версии 2.1x |
| | 3.11 | 1991 | существенно расширенные возможности NetWare 386; стала первой сетевой операционной системой, обеспечивающей доступ к сетевым ресурсам с рабочих станций DOS, Windows, OS/2, UNIX и Macintosh |
| NetWare SFT Level III отказоустойчивая III-го уровня | 3.11 | 1993 | разработана специально для использования в системах, требующих наивысшего уровня надежности; SFT III обеспечивает работу двух серверов в «зеркальном» режиме, при этом один из серверов всегда является активным, а второй находится в горячем резерве, обеспечивая у себя такое же состояние памяти и дисков, как и у основного сервера |
| NetWare | 4.0 | 1993 | система для построения вычислительных сетей «масштаба предприятия»; во многих отношениях революционно новый продукт. Основным нововведением является служба каталогов NetWare Directory Services (NDS) — распределенная база данных всех сетевых ресурсов в данной ЛВС |
| | 3.12 | 1993 | усовершенствованный вариант самой популярной сетевой ОС фирмы Novell — NetWare v3.11; убраны ошибки предыдущей версии, добавлены возможности клиентов (Macintosh, DOS VLM). |
| | 3.20 | | официально является последней версией NetWare 386, не поддерживается с 2002г. |
| | 4.10 | 1994 | существенно улучшена служба NDS |
| | 4.20 | 1998 | последняя версия 4.x, исправлены все обнаруженные ошибки |

Окончание табл. 2.1

| Название | Версия | Год выпуска | Особенности |
|---|---|---|---|
| | 5.0 | 1999 | поддержка мультипроцессоров, виртуальная память, универсальная система хранения файлов любого размера, поддержка Java, средства для WWW, реализация «чистого» TCP/IP |

В настоящее время (2004 г.) выпущена версия NetWare 6.5. Основное направление развития этой ОС — интеграция с сетевыми продуктами других фирм, поддержка интернет-технологий.

Серверные платформы: компьютеры на основе процессоров Intel, рабочие станции RS/6000 компании IBM под управлением операционной системы AIX с помощью продукта NetWare for UNIX.

Клиентские платформы: DOS, Macintosh, OS/2, UNIX, Windows 9x/NT/2000/XP. Встроена справочная служба NetWare Directory Services (NDS), поддерживающая централизованное управление, распределенную, полностью реплицируемую, автоматически синхронизируемую и обладающую отличной масштабируемостью.

Служба NDS фактически является лидером среди подобных продуктов других фирм.

Поддерживаемые сетевые протоколы: TCP/IP, IPX/SPX, NetBIOS, Appletalk.

Поддержка удаленных пользователей: ISDN, коммутируемые телефонные линии, frame relay, X.25 — с помощью продукта NetWare Connect (поставляется отдельно).

Безопасность: аутентификация с помощью открытых ключей метода шифрования RSA; сертифицирована по уровню C2. Хороший сервер коммуникаций

Встроенная функция компрессии диска.

Общий итог — очень эффективный и надежный файл-сервер, а также средство (на основе NDS) для интеграции сетевых ОС. В 2002г. Novell приобрела фирмы SUSE и Ximian выпускающие соответственно Linux-системы для серверных и настольных приложений. В этих действиях усматривается прицел потеснить Microsoft на рынке настольных пользовательских операционных систем. Остается только пожелать успеха фирме Novell в продвижении своих продуктов.



*Windows NT/2000/XP*

В конце 88-го года Microsoft поручила Дэвиду Катлеру (David Cutler) возглавить новый проект в области программного обеспечения: создать новую ОС фирмы Microsoft для 90-х годов. (Дэвид Катлер — главный консультант фирмы DEC, который 17 лет проработал там, разрабатывая ОС и компиляторы: VAX/VMS, ОС для MicroVAX I, OS RSX-11M, компиляторы VAX PL/1, VAX C). Он собрал команду инженеров для разработки ОС новой технологии (New Technology — NT).

Первоначально планировалось разработать NT с пользовательским и программным (API) интерфейсами в стиле OS/2, однако OS/2 плохо продавалась, а Windows 3.0 имела большой и постоянный успех на рынке. Увидев рыночные ориентиры и сложности, связанные с развитием и поддержкой двух несовместимых систем, Microsoft решила изменить свой курс и направить своих инженеров в сторону стратегии единой цельной операционной системы. Эта стратегия состоит в том, чтобы разрабатывать семейство базирующихся на Windows операционных систем, которые охватывали бы множество типов компьютеров, от самых маленьких ноутбуков до самых больших мультипроцессорных рабочих станций. Windows NT, как было названо следующее поколение Windows-систем, занимает самое высокое место в семействе Windows. Она поддерживает графический интерфейс (GUI) пользователя Windows, а также является первой базирующейся на Windows операционной системой фирмы Microsoft, поддерживающей Win32 API, 32-х битный программный интерфейс для разработки новых приложений. Win32 API делает доступными для приложений улучшенные свойства ОС, такие как многонитевые процессы, синхронизацию, безопасность, управление объектами.

В июле 1993 года появились первые ОС семейства NT — Windows NT 3.1 и Windows NT Advanced Server 3.1.

Серверные платформы: компьютеры на базе процессоров Intel, PowerPC, DEC Alpha, MIPS

Клиентские платформы: DOS, OS/2, Windows, Macintosh

Организация одноранговой сети возможна с помощью Windows NT Workstation и Windows 95/98/ME.

Windows NT Server представляет собой отличный сервер приложений: он поддерживает вытесняющую многозадачность, виртуальную память и симметричное мультипроцессирование, а также прикладные среды DOS, Windows, OS/2, POSIX

Справочные службы: доменная для управления учетной информацией пользователей (Windows NT Domain Directory service), справочные службы имен WINS и DNS

Хорошая поддержка совместной работы с сетями NetWare: поставляется клиентская часть (редиректор) для сервера NetWare (версий 3.x и 4.x в режиме эмуляции 3.x, справочная служба NDS поддерживается, начиная с версии 4.0), выполненная в виде шлюза в Windows NT Server или как отдельная компонента для Windows NT Workstation; недавно Microsoft объявила о выпуске серверной части NetWare как оболочки для Windows NT Server.

Поддерживаемые сетевые протоколы: TCP/IP, IPX/SPX, NetBEUI, Appletalk.

Поддержка удаленных пользователей: ISDN, коммутируемые телефонные линии, frame relay, X.25 — с помощью встроенной подсистемы Remote Access Server (RAS).

Служба безопасности: мощная, использует избирательные права доступа и доверительные отношения между доменами; узлы сети, основанные на Windows NT Server, сертифицированы по уровню C2 Простота установки и обслуживания.

Отличная масштабируемость. В Windows 2000 добавлена служба каталогов Active Directory. Возможность терминальной работы. Следует отметить, что, начиная с версии 2000, прекратила свое развитие линия Windows95/98/ME.

В последующих версиях линии Windows NT, XP (Home Edition и Professional Edition), встроена поддержка новейшего оборудования, облегчена работа пользователя с сетью.

В целом набор возможностей, предлагаемых Windows NT, может удовлетворить запросы большинства пользователей.



И в последние годы NT стало уверенно теснить Novell Netware на рынке серверов локальных сетей. Тем не менее в области службы каталогов или справочной службы Active Directory еще очень далеко до NDS.

## 2.3 Стандарты сетей ЭВМ

Важным аспектом современных сетей является всеобъемлющая стандартизация, так как ВС являются объединением самых разнообразных технологий, программного и аппаратного обеспечения, зачастую производимого разными фирмами, то проблема совместимости является первостепенной. Стандарты приходят из разных источников:

- отдельные фирмы являющиеся передовыми на данном рынке (IBM PC);
- специальные комитеты и объединения (ATM, Fast Ethernet Alliance);
- национальные стандарты (IEEE, ANSI);
- международные стандарты (ISO, ITU).

В мире существует большое количество производителей сетей, каждый из которых имеет свои представления о способах реализации различных функций. Без координации их действий наступила бы полная неразбериха, и пользователи сетей не смогли бы с ней справиться. Единственным способом борьбы с хаосом является достижение согласия по определенным вопросам на основе сетевых стандартов.

Стандарты не только обеспечивают возможность общения различных компьютеров, но также расширяют рынок для продукции, придерживающейся стандарта, что приводит к массовому выпуску совместимой друг с другом аппаратуры, очень широкомасштабной интеграции (VLSI, Very Large Scale Integration), удешевлению производства и, следовательно, к еще большему притоку потребителей на этот рынок.

Стандарты делятся на две категории: *de facto* и *de jure*. Стандарты de facto (лат. «фактические») установились сами собой, без какого-либо предварительного плана. Так, например, персональные компьютеры IBM PC и их преемники стали стандартом de facto для небольших офисных компьютеров, поскольку

ку десятки производителей решили довольно детально копировать машины фирмы IBM. Система UNIX стала стандартом de facto для операционных систем университетских компьютеров.

Стандарты de jure (лат. «юридические»), напротив, являются формальными, легитимными стандартами, принятыми некими авторитетными организациями по стандартизации. Международные организации по стандартизации обычно делятся на два класса: созданные на основе межправительственных договоров и добровольные организации. В области сетевых компьютерных стандартов существуют несколько организаций каждого типа.

*Кто есть кто в мире телекоммуникаций*

Официальный статус телефонных компаний мира в разных странах различен. На одном полюсе находятся Соединенные Штаты, в которых имеется 1500 отдельных частных телефонных компаний. До разделения в 1984 году AT&T, самая большая на тот момент корпорация в мире, полностью доминировала в этом рыночном секторе. Ее услугами пользовались около 80 % абонентов Америки, что покрывало примерно половину территории страны, тогда как все остальные компании поставляли услуги остальным клиентам (в основном провинциям). После разделения AT&T продолжает предоставлять услуги по междугородной связи, теперь уже конкурируя с другими компаниями. Семь региональных телефонных компаний (RBOC, Regional Bell Operating Company), на которые была разбита AT&T, и 1500 независимых компаний предоставляют местные телефонные услуги, а также услуги сотовой связи. Благодаря постоянному процессу объединения и раскола компаний, эта отрасль промышленности находится в состоянии непрерывного изменения.

Компании в США, предоставляющие общественные услуги в области связи, называются *операторами связи общего пользования* (common carriers) или просто *операторами связи***.** Предоставляемые ими услуги и цены на эти услуги описываются в документе, называющемся *тарифом* (tariff), который должен быть одобрен Федеральной комиссией связи США (FCC, Federal Communication Commission), если речь идет о международной связи и связи между штатами, и комиссиями по коммунальным услугам штатов применительно к связи внутри штата.



На другом полюсе располагаются страны, в которых правительство обладает полной монополией на все средства связи, включая почту, телеграф, телефон, а часто также и радио, и телевидение. В эту категорию попадает большая часть мира. В некоторых случаях владельцем средств связи выступает национализированная компания, в других им является просто особая ветвь правительства, обычно называемая Министерством связи или PTT (Postal Telegraph and Telephone administration — Управление почтово-телеграфной и телефонной связи). Сейчас во всем мире наблюдается тенденция перехода от государственной монополии к либерализации и конкуренции. Большинство европейских стран уже передало целиком или частично эту отрасль в руки независимых собственников, но кое-где процесс пока еще идет слишком медленно.

При таком разнообразии поставщиков услуг, очевидно, имеется необходимость в обеспечении совместимости во всемирном масштабе. Совместимости, гарантирующей пользователям (и компьютерам) разных стран возможность связаться друг с другом. На самом деле, эта потребность существовала уже очень давно. В 1865 году представители многих европейских государств собрались, чтобы сформировать союз, который явился предшественником сегодняшнего Международного союза телекоммуникаций (ITU, International Telecommunications Union). Задачей этого союза стала стандартизация международных средств связи. В то время еще нечего было стандартизировать, кроме телеграфа, но уже тогда было ясно, что если половина стран будет использовать азбуку Морзе, а другая половина — какой-либо другой код, то возникнут проблемы. С появлением международной телефонной связи ITU занялся также разработкой стандартов в области телефонии. В 1947 году международный союз телекоммуникаций вошел в состав учреждений Организации Объединенных Наций.

Международный союз телекоммуникаций состоит из трех главных секторов:

- сектор радиосвязи (ITU-R);
- сектор стандартизации телекоммуникаций (ITU-T);
- сектор развития (ITU-D).

Сектор радиосвязи (ITU-R) занимается вопросами предоставления радиочастот конкурирующим компаниям во всем мире. Нас же в первую очередь интересует сектор стандартизации телекоммуникаций (ITU-T), рассматривающий вопросы, связанные с телефонными системами и системами передачи данных. С 1956 по 1993 год ITU-T назывался Консультативным комитетом по международной телефонной и телеграфной связи (CCITT, Comite Consultatif International Telegraphique et Telephonique). 1 марта 1993 года CCITT был реорганизован и переименован в ITU-T, что больше отражало роль этой организации. Как CCITT, так и ITU-T выпускали рекомендации в области телефонных систем и систем передачи данных. Рекомендации CCITT встречаются еще довольно часто, как, например, CCITT X.25, хотя с 1993 года рекомендации носят эмблему ITU-T.

В ITU-T входят представители

- национальных административных структур;
- соответствующего промышленного сектора;
- организаций;
- регулятивных органов.

В ITU-T входят примерно 200 представителей различных ведомств, включая практически все организации ООН. В США отсутствует PTT, но кто-то же должен представлять государственные структуры. Эта роль выпала Государственному департаменту. Возможно, он участвует в решении вопросов взаимодействия ITU-T с другими государствами, что коррелирует с собственным направлением деятельности Госдепартамента. В состав ITU-T входит примерно 500 представителей соответствующего промышленного сектора, среди которых есть и телефонные компании (например, AT&T, Vodafone, WorldCom), и производители телекоммуникационной аппаратуры (например, Cisco, Nokia, Nortel), и производители компьютеров (например, Compaq, Sun, Toshiba), производители микросхем (например, Intel, Motorola, TI), медиа-компании (например, AOL Time Warner, CBS, Sony) и другие заинтересованные компании (например, Boeing, Samsung, Xerox). Различные некоммерческие научные организации и промышленные консорциумы (например, IFIP и IATA) также входят в состав представителей промышленного сектора. Другую группу составляют представители относительно небольших организаций, заинтересованных в конкретных направлениях стандартизации. Регулятивные органы наблюдают за развитием телекоммуникационного бизнеса. Среди них, например, Федеральная комиссия связи США (FCC).



Задачей ITU-T является разработка технических рекомендаций в области телефонии, телеграфа и интерфейсов передачи данных. Подобные рекомендации часто становятся международными стандартами, как, например, V.24 (известный также как EIA RS-232), описывающий расположение и назначение контактов разъема, используемого в большинстве асинхронных терминалов и внешних модемов.

Следует заметить, что рекомендации ITU-T являются всего лишь предложениями, которые правительство любой страны может принять или проигнорировать. На практике никто не может помешать стране принять свой собственный телефонный стандарт, отличающийся от всех других, однако тем самым эта страна сама себя отрежет от всего мира. Возможно, такой подход будет работать в Северной Корее, но во всем остальном мире это будет настоящей проблемой. Так что использование названия «рекомендации» для стандартов ITU-T можно считать выдумкой для умиротворения националистов.

Собственно работа в ITU-T осуществляется исследовательскими группами (Study Groups), состав которых часто достигает 400 человек. В настоящее время (2003г) работают 14 таких групп, занимающихся самыми разными вопросами, начиная от концепции оплаты телефонных услуг и заканчивая службами мультимедиа. Чтобы сделать возможной какую-либо работу, исследовательские группы разделяются на рабочие группы (Working Parties), подразделяющиеся в свою очередь на экспертные команды (Expert Teams), которые также делятся на группы.

Производительность ITU-T — около 3000 рекомендаций в год, что составляет 60 000 страниц печатного текста. Многие из них широко применяются на практике. Например, популярный стандарт V-90 для модемов со скоростью 56 Кбит/с является ничем иным, как рекомендацией ITU.

По мере того, как телекоммуникации завершают начатый ими в 80-е годы переход от национальных систем к глобальным, стандарты становятся все более важным аспектом их развития,

и все большее число организаций желает принять участие в их формировании.

*Кто есть кто в мире международных стандартов*

Международные стандарты разрабатываются Международной организацией по стандартизации (International Organization for Standardization, ISO), добровольной организацией, созданной в 1946 году. В нее входят национальные организации по стандартизации из 89 стран. Среди ее членов такие организации, как ANSI (США), BSI (Великобритания), AFNOR (Франция), DIN (Германия) и др.

ISO выпускает стандарты, касающиеся широкого спектра вопросов, начиная от болтов и гаек (буквально) до покраски телефонных столбов (не говоря уж о стандартах на какао-бобы (ISO 2451), рыболовные сети (ISO 1530), женское нижнее белье (ISO 4416). Сложно придумать такую вещь, для которой не существовало бы стандарта ISO). К настоящему времени выпущено более 13 000 стандартов, включая стандарты OSI. В ISO входят почти 200 технических комитетов (Technical Committee, TC), нумеруемых последовательно, по мере их создания, каждый из которых занимается своим отдельным вопросом. Так, например, TC1 занимается болтами и гайками (стандартизацией диаметра и шага резьбы). Компьютеры и обработка информации входят в сферу деятельности TC97. Каждый технический комитет делится на подкомитеты (subcommittee, SC), которые, в свою очередь, состоят из рабочих групп (working group, WG).

Основная работа проводится в рабочих группах, в которые входит более 100 000 добровольцев по всему миру. Многие из этих «добровольцев» работают по найму в компаниях, чьи продукты должны быть стандартизованы. Другие являются государственными служащими, пытающимися сделать свой национальный стандарт международным. Ученые эксперты также принимают активное участие во многих рабочих группах.

В области телекоммуникационных стандартов ISO и ITU-T часто сотрудничают (ISO является членом ITU-T), чтобы не допустить возникновения двух несовместимых международных стандартов.



Представителем США в ISO является Национальный институт стандартизации США (ANSI, American National Standards Institute), который, несмотря на свое название, является частной неправительственной некоммерческой организацией. Ее членами являются производители, операторы связи и другие заинтересованные стороны. Стандарты ANSI часто принимаются ISO в качестве международных.

Для принятия стандартов Международной организацией по стандартизации ISO разработана процедура, позволяющая добиться принятия мнения, одобренного максимально возможным количеством сторон. Процесс начинается, когда одна из национальных организаций по стандартизации чувствует потребность в появлении международного стандарта в определенной области. Тогда формируется рабочая группа, которая вырабатывает предварительный план (Committee Draft, CD). Этот набросок передается всем остальным членам технического комитета. На критику проекта отводится шесть месяцев. Если проект получает одобрение значительного большинства, то откорректированный документ получает название чернового международного стандарта (DIS, Draft International Standard); после этого он опять циркулирует в различных группах, где за него голосуют и снабжают его комментариями. На основании этого подготавливается, утверждается и публикуется окончательный документ, называемый международным стандартом (IS, International Standard). В некоторых случаях черновым вариантам, CD или DIS приходится проходить через многократные изменения и голосования, пока они не наберут достаточного количества голосов. Этот процесс может длиться несколько лет.

Национальный институт стандартов и технологий США (NIST, National Institute of Standards and Technology) является агентством Министерства торговли США (U.S. Dept. of Commerce). Ранее он назывался Национальным бюро стандартов (National Bureau of Standards). Он выпускает стандарты, обязательные для закупок, проводимых правительством США, кроме закупок Министерства обороны, у которого имеются свои стандарты.

Одним из основных игроков на поле стандартизации является Институт инженеров по электротехнике и электронике (IEEE, Institute of Electrical and Electronics Engineers) — круп-

нейшая профессиональная организация в мире. Помимо выпуска ряда журналов и организации разнообразных конференций, IEEE также разрабатывает стандарты в области электротехники и электроники. Например, комитет IEEE 802 выпустил ряд ключевых стандартов в области локальных компьютерных сетей.

Таблица 2.2

| Номер | Тема разработок |
|---|---|
| 802.1 | Общее представление и архитектура ЛВС |
| 802.2 v | Управление логическим каналом |
| 802.3 * | Ethernet |
| 802.4 v | Маркерная шина (одно время использовалась в промышленных сетях) |
| 802.5 | Маркерное кольцо (вклад фирмы IBM в технологии ЛВС) |
| 802.6 v | Двойная двунаправленная шина (ранние региональные сети) |
| 802.2 v | Техническая консультативная группа по широкополосным технологиям |
| 802.8 + | Техническая консультативная группа по оптоволоконным технологиям |
| 802.9 v | Изохронные ЛВС (для приложений реального времени) |
| 802.10 v | Виртуальные ЛВС и защита информации |
| 802.11 * | Беспроводные ЛВС |
| 802.12 v | Приоритеты запросов (для AnyLAN фирмы Hewlett-Packard) |
| 802.13 | Почему-то этот номер никто не выбрал. |
| 802.14 v | Кабельные модемы (рабочая группа распалась: в области кабельных модемов ее опередил промышленный консорциум) |
| 802.15 * | Персональные сети (Bluetooth) |
| 802.16 * | Широкополосные беспроводные ЛВС |
| 802.17 | Гибкая технология пакетного кольца |

Реальная работа проводится, как правило, внутри рабочих групп, которые перечислены в таблице 2.2 (наиболее важные группы отмечены звездочкой (*); помеченные галочкой (v) бездействуют; отмеченные крестиком (+) самоликвидировались за ненадобностью). Рабочие группы комитета 802 никогда не считались преуспевающими. Однако стремительный взлет популярности стандартов типа 802.3 и 802.11 изменил ситуацию.



*Кто есть кто в мире стандартов Интернета*

Всемирная сеть Интернет имеет свой механизм стандартизации, значительно отличающийся от ITU-T и ISO. В двух словах основное отличие заключается в том, что сотрудники ITU и ISO носят деловые костюмы, тогда как стандарты Интернета разрабатывают в основном люди в джинсах.

На совещания ITU-T и ISO собираются администраторы корпораций и государственные гражданские служащие, для которых стандартизация является работой. Для людей Интернета, напротив, анархия является делом принципа, однако когда сотни миллионов делают какое-то общее дело, иногда им все же приходится о чем-то договариваться, чтобы хоть что-то работало. Волей-неволей стандарты оказываются необходимыми.

Когда была запущена сеть ARPANET, Министерство обороны США создало неофициальный комитет для наблюдения за сетью. В 1983 году этот комитет был переименован в Совет по деятельности Интернета (Internet Activities Board, IAB). Перед советом были поставлены несколько расширенные задачи, а именно: удерживать исследователей, включенных в проекты ARPANET и Интернет, в более или менее одном направлении. Значение сокращения IAB было затем изменено на Совет по архитектуре Интернета (Internet Architecture Board).

Каждый из приблизительно десяти членов IAB возглавлял специальную комиссию по отдельному важному вопросу. Совет по архитектуре Интернета собирался несколько раз в год для обсуждения результатов работы и представления отчета Министерству обороны и NSF, которые в то время осуществляли основное финансирование в этой области. Когда требовался какой-либо стандарт (например, новый алгоритм маршрутизации), члены совета прорабатывали этот вопрос, после чего объявляли об изменениях аспирантам, занимавшимся реализацией программного обеспечения сетей. Стандарты оформлялись в виде набора технических отчетов, называемых RFC (Requests for Comments). RFC доступны в Интернете для всех желающих (www.ietf.org/rfc). Они пронумерованы в хронологическом по-

рядке их создания. На сегодняшний день существует около 3000 этих документов. На некоторые из них мы будем ссылаться в главах 4 и 5 пособия.

К 1989 году Интернет вырос настолько, что подобный неформальный подход к его стандартам перестал работать. К тому моменту многие производители предлагали продукцию на основе протокола TCP/IP и не хотели ее менять просто потому, что десятку исследователей пришла в головы одна хорошая идея. Летом 1989 года IAB был снова реорганизован, исследователи были переведены в группу исследования Интернета (Internet Research Task Force, IRTF), подконтрольную IAB, и в группу проектирования Интернета (Internet Engineering Task Force, IETF). В совете IAB появились люди, представляющие более широкий спектр организаций, чем исследовательское сообщество. Вначале это была группа, в которой члены работали в течение двух лет, после чего сами назначали своих преемников. Затем было создано Общество Интернета (Internet Society), в которое вошли люди, заинтересованные в Интернете. Таким образом, Интернет-сообщество в каком-то смысле сравнимо с Ассоциацией по вычислительной технике (ACM, Association for Computing Machinery) или IEEE. Оно управляется избираемыми доверенными лицами, которые утверждают состав IAB.

Идея этого разделения заключалась в том, чтобы сосредоточить IRTF на долгосрочных исследованиях, а IETF — на краткосрочных инженерных вопросах. Проблемная группа IETF была разделена на рабочие группы, каждая из которых решала свою задачу. Первое время председатели рабочих групп встречались друг с другом в составе руководящего комитета для координации совместных исследовательских усилий. Рабочие группы занимались такими вопросами, как новые приложения, информация для пользователей, OSI-интеграция, маршрутизация и адресация, безопасность, управление сетью и стандарты. В конце концов было сформировано так много рабочих групп (более 70), что их объединили по областям, после чего в руководящем комитете стали собираться председатели областей.

Кроме того, был принят более формальный процесс стандартизации по аналогии с процедурой, принятой в ISO. Чтобы стать предлагаемым стандартом, основная идея должна быть



полностью изложена в RFC и должна представлять достаточный интерес, гарантирующий ее рассмотрение. Затем, чтобы стать проектом стандарта, должна быть создана работающая реализация, которую нужно тщательно протестировать минимум двумя независимыми сайтами в течение 4 месяцев. Если IAB уверен, что идея здравая и программное обеспечение работает, он может объявить RFC стандартом Интернета. Некоторые стандарты Интернета стали стандартами Министерства обороны США (MIL-STD), что сделало их обязательными к применению поставщиками министерства. Дэвид Кларк (David Clark) высказал замечание, ставшее ныне популярным, о стандартизации Интернета, состоящей из «грубого консенсуса и работающей программы».

## 2.4 Модели сетевого взаимодействия

### 2.4.1 Модель OSI (ВОС)

В начале 80-х годов ряд международных организаций по стандартизации (ISO, ITU-T и некоторые другие) разработали модель, которая сыграла значительную роль в развитии сетей. Это модель OSI (open system interconnection) или ВОС (взаимодействия открытых систем). При ее создании использовался опыт, полученный при создании сетей, в основном глобальных, в 70-е годы. В модели (рис. 2.6) семь уровней, каждый из которых имеет дело с одним определенным аспектом взаимодействия сетевых устройств.

Три нижних уровня — физический, канальный и сетевой — называются *сетезависимыми*, то есть протоколы этих уровней тесно связаны с технической реализацией сети и используемым коммуникационным оборудованием. Например, переход от технологии Ethernet к технологии FDDI означает полную смену протоколов физического и канального уровней во всех узлах сети. Три верхних уровня — прикладной, представительный и сеансовый — ориентированы на приложения и мало зависят от технических особенностей построения сети. Транспортный уровень является промежуточным, он скрывает технические детали реализации от вышестоящих уровней. Данная модель описывает только системные средства взаимодействия, реализуемые опе-

рационной системой, системными утилитами, системными аппаратными средствами.

| | | |
|---|---|---|
| Прикладной уровень | 7 | Взаимодействие с прикладными программами. Предоставление стандартных сервисов (файловый, приложений, сообщений, печати и др.) |
| Представительный уровень | 6 | Согласование форматов данных различных систем. Соглашение об именах, кодирование символьной информации, шифрование. |
| Сеансовый уровень | 5 | Обеспечивает управление диалогом.<br>Установление, завершение соединения. Повторная передача при возникновении ошибок. |
| Транспортный уровень | 4 | Предоставляет услуги транспортировки данных с заданным уровнем качества. Скрывает детали передачи данных от вышестоящих уровней. Разбиение сообщения на сегменты при передаче и сборка при приеме. |
| Сетевой уровень | 3 | Объединение нескольких сетей в одну единую сеть.<br>Задача маршрутизации. Сервис шлюзов между сетями с разными технологиями. Каналы и датаграммы. |
| Канальный уровень | 2 | Передача кадров (групп битов).<br>Доступ к среде (MAC - подуровень).<br>Контроль логического соединения (LLC - подуровень) |
| Физический уровень | 1 | Передача битов по физическим каналам связи.<br>Характеристики сред передачи данных. |

Рисунок 2.6 — Модель взаимодействия открытых систем

Однако, приложения могут реализовывать собственные протоколы, обращаясь к системным средствам. Поэтому следует различать прикладной уровень и уровень взаимодействия приложений.

Модель ВОС предполагает, что на двух взаимодействующих системах программное и аппаратное обеспечение должно быть построено по модульному принципу. Каждый модуль соответствует определенному уровню модели ВОС. Каждый из модулей взаимодействует только со своими «соседями», расположенными снизу или сверху. Взаимодействие модулей одной системы строго регламентировано. Вышестоящий модуль использует сервис, предоставляемый нижестоящим модулем. Между соседними модулями должны быть четко оговоренные правила взаимодействия — интерфейсы. *Интерфейс* — это все,



что требуется «знать» одному модулю о другом. Внутренняя реализация не берется во внимание.

Таким образом, мы можем изменять реализацию какого-либо модуля на более совершенную, но это изменение не затронет другие модули, так как интерфейс или «контракт» между модулями не изменился. Другими словами, модуль (программный или аппаратный) инкапсулирует свою реализацию. Модули передают друг другу блоки данных, обрабатываемых на каждом уровне по-своему.

В целом весь набор модулей одной системы (компьютера) решает задачу по передаче данных (сообщений) другой системе. Переданное сообщение должно быть правильно понято на другой стороне. Поэтому модули формируют сообщения по специальным правилам, называемым *протоколами*. Интерфейс и протокол суть взаимозаменяемые понятия, но интерфейс — это взаимодействие более тесное между модулями одной системы, находящимися на разных уровнях модели, а протокол — удаленное взаимодействие между модулями двух разных систем, но находящихся на одном уровне модели (рис. 2.7).

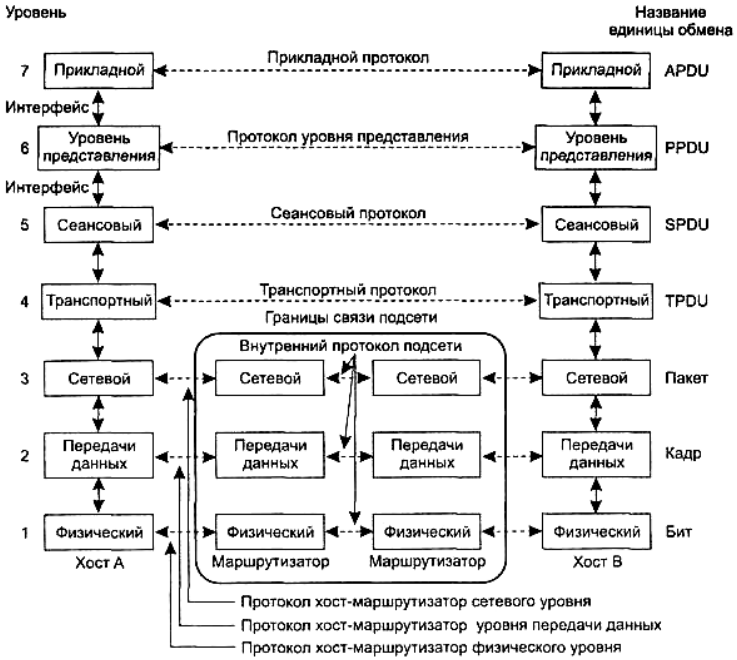



Рисунок 2.7 — Взаимодействие открытых систем в соответствии с моделью OSI

### 2.4.2 Модель DoD. Сравнение с моделью OSI

Модель OSI не является единственной моделью сетевого взаимодействия. Стек протоколов TCP/IP был разработан ранее и принцип модульности и разбиения на уровни также использовался.

Модель Департамента защиты (рис. 2.8) (Department of Defense — DoD) заведомо не определяет стандарты физического уровня. Любая технология может быть использована для транспортировки пакетов TCP/IP, если удастся согласовать интерфейсы. Такая гибкость TCP/IP способствовала широкому распространению и популярности этого стека.




Рисунок 2.8 — Сравнение моделей ISO и TCP/IP (DoD)

Эталонная модель OSI была разработана *прежде,* чем были изобретены протоколы для нее, то есть прежде, чем она получила практическое применение. Такая последовательность событий означает, что эта модель не была настроена на какой-то конкретный набор протоколов, что сделало ее универсальной. Обратной стороной такого порядка действий было то, что у разработчиков было мало опыта в данной области и не было четкого представления о том, какие функции должен выполнять каждый уровень.

Например, уровень передачи данных изначально работал только в сетях с передачей от узла к узлу. С появлением широковещательных сетей в модель потребовалось ввести новый подуровень. Когда же на базе модели OSI начали строить реальные сети с использованием существующих протоколов, обнаружилось, что они не соответствуют требуемым спецификациям

служб. Поэтому в модель пришлось добавить подуровни для устранения несоответствия. Наконец, изначально ожидалось, что в каждой стране будет одна сеть, управляемая правительством и использующая протоколы OSI, поэтому никто и не думал об объединении различных сетей. В действительности все оказалось не так.

С моделью TCP/IP было все наоборот: сначала появились протоколы, а уже затем была создана модель, описывающая существующие протоколы. Таким образом, не было проблемы с соответствием протоколов модели. Они ей соответствовали прекрасно. Единственной проблемой было то, что *модель* не соответствовала никаким другим стекам протоколов. В результате она не использовалась для описания каких-нибудь других сетей, отличных от TCP/IP.

Если взглянуть на эти две модели поближе, то прежде всего обратит на себя внимание различие в количестве уровней: в модели OSI семь уровней, в модели TCP/IP — четыре. В обеих моделях имеются межсетевой, транспортный и прикладной уровни, а остальные уровни различные.

Еще одно различие между моделями лежит в сфере возможности использования связи на основе соединений и связи без установления соединения. Модель OSI на сетевом уровне поддерживает оба типа связи, а на транспортном уровне — только связь на основе соединений (поскольку транспортные службы являются видимыми для пользователя). В модели TCP/IP на сетевом уровне есть только один режим связи (без установления соединения), но на транспортном уровне он поддерживает оба режима, предоставляя пользователям выбор. Этот выбор особенно важен для простых протоколов «запрос — ответ».

При создании модели OSI больше внимания уделялось абстрактному описанию сервисов, интерфейсов и протоколов, что бы их можно было легко заменять в дальнейшем. Это напоминает объектно-ориентированный подход в программировании.

TCP/IP создавался в некоторой степени стихийно, под девизом: «Только бы оно заработало». Понятия сервис, интерфейс и протокол там не столь четко разделены и это в настоящее время



является серьезным препятствием для дальнейшего развития TCP/IP.

Если модель OSI в силу своей универсальности может быть использована для описания самых разнообразных протоколов, то модель DoD может использоваться только для описания стека протоколов TCP/IP и ни для чего больше. Поэтому модель OSI используется специалистами во всем мире, как некий единый стандарт описания взаимодействующих систем.

# 3 ОПИСАНИЕ СЕРВИСА В МОДЕЛИ OSI

## 3.1 Общие сведения

Прежде всего заметим, что при рассмотрении технологии передачи информации мы будем опираться на понятие взаимодействия процессов. То есть в сети ЭВМ источниками и получателями информации являются именно прикладные процессы (может быть, имеющие разную природу), а не оборудование, пользователи и т.п. Функции прикладных процессов, которые связаны с обеспечением взаимодействия, называются *прикладными объектами*.

Известно, что для непосредственной передачи данных между системами необходима некоторая физическая среда.

Таким образом, эталонная модель взаимодействия открытых систем (ВОС), с точки зрения разработчика ПО, основана на четырех элементах (рис. 3.1): открытых системах; прикладных объектах, существующих в рамках ВОС; соединениях, которые связывают прикладные объекты и позволяют им обмениваться информацией; физической среде для ВОС.

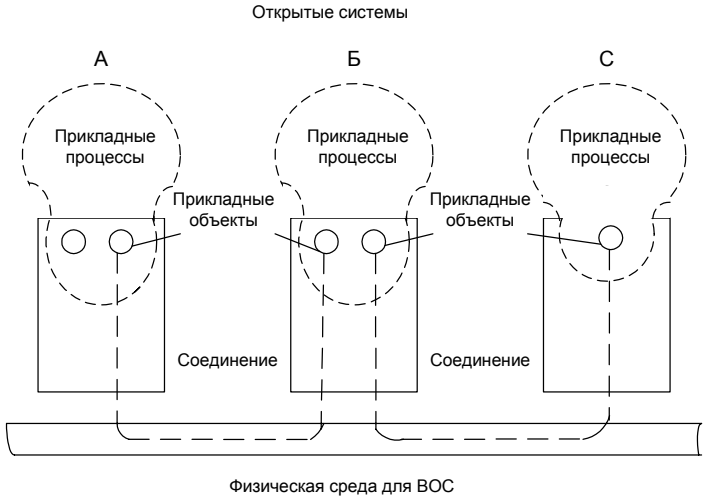



Рисунок 3.1 — Основные элементы эталонной модели ВОС

Замечание: термин «соединение» в данном контексте имеет широкое толкование. Его следует понимать как любую кооперацию прикладных объектов вообще. Такая кооперация может принимать самые разнообразные формы. К ним можно отнести:

- обмен информацией и синхронизация действий между прикладными процессами;
- представление данных, относящееся ко всем аспектам создания и поддержания описаний и преобразований данных;
- управление ресурсами, с помощью которых инициируются прикладные процессы ВОС;
- обеспечение целостности и сохранности данных во время функционирования ВОС.



Из этого следует то, что модель ВОС не ограничивается только передачей информацией между системами, но и затрагивает вопросы взаимодействия для решения общих задач.

Функции взаимосвязи, образующие открытую систему, в своей совокупности чрезвычайно сложны. Поэтому в модели ВОС каждая открытая система представляется в виде иерархически расположенных подсистем (рис. 3.2). Подсистемы, выполняющие схожие наборы функций взаимосвязи, образуют уровень. Для локализации функции используется понятие объекта. Таким образом, подсистема N-уровня (N-подсистема) состоит из одного или нескольких N-объектов.

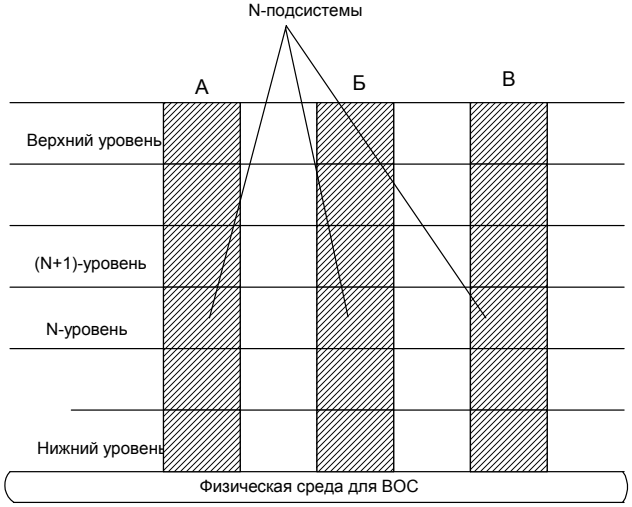



Рисунок 3.2 — Организация уровней во взаимосвязанных открытых системах

Полная схема взаимодействия выглядит следующим образом: за исключением верхнего уровня каждый N-уровень обеспечивает для (N+1)-объектов в (N+1)-уровне N-услуги (так называемые службы). Совокупность услуг N-уровня называется сервисом N-уровня, или N-сервисом. Когда некоторый N-объект не может сам обеспечить полную поддержку услуги, запрашиваемую некоторым (N+1)-объектом, он вызывает другие N-объекты для помощи в обеспечении запроса на обслуживание. Для выполнения такого взаимодействия N-объекты любого уровня, исключая нижний, связываются посредством набора

услуг, предоставляемых (N-1)-уровнем (рис. 3.3). Услуги N-уровня предоставляются (N+1)-уровню путем выполнения N-функций внутри N-уровня, а также использования услуг, получаемых от (N-1)-уровня.

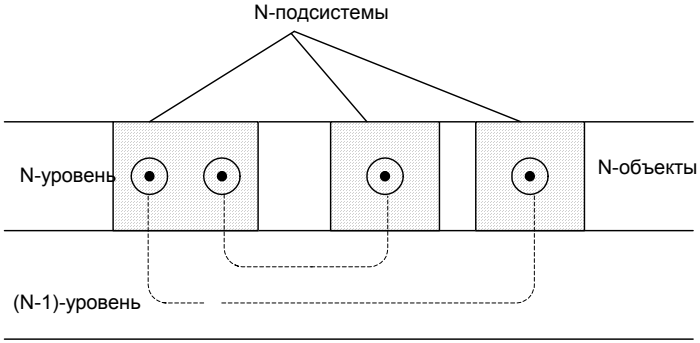



Рисунок 3.3 — Связь N-объектов через (N-1)-уровень

Взаимосвязь между N-объектами осуществляется под управлением одного или нескольких N-протоколов (рис. 3.4).

Объекты (N+1) могут связываться между собой только с помощью услуг, предоставляемых N-уровнем. Возможны случаи, когда услуги, предоставляемые N-уровнем, не позволяют обеспечить прямое взаимодействие между всеми N-объектами. В этих случаях взаимодействие может осуществиться, если некоторый из (N+1)-объектов выполнит функцию ретранслятора (рис. 3.5). Тот факт, что связь ретранслируется цепочкой (N+1)-объектов, неизвестен ни N-уровню, ни (N+2)-уровню.

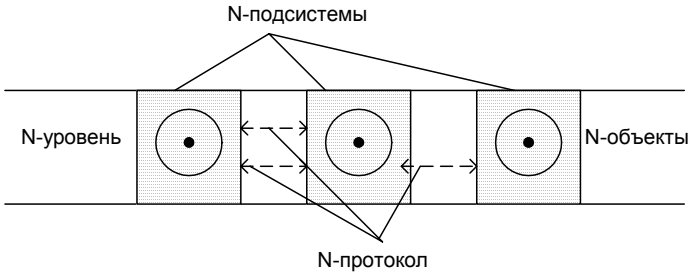



Рисунок 3.4 — Место N-протоколов между N-объектами



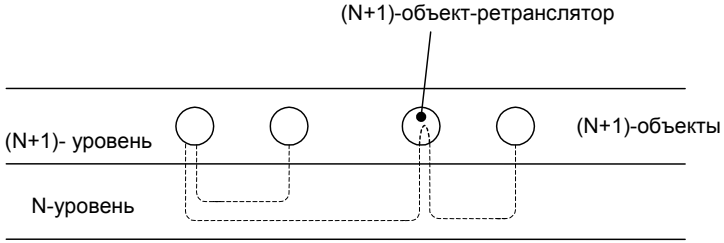



Рисунок 3.5 — Связь (N+1)-объектов через (N+1)-объект-ретранслятор

## 3.2 Функции уровней

Стандарт определяет конкретное содержание модели ВОС, регламентируя число уровней взаимосвязи и их функции. Эталонная модель ВОС содержит семь уровней:

- прикладной (уровень 7);
- представительный (уровень 6);
- сеансовый (уровень 5);
- транспортный (уровень 4);
- сетевой (уровень 3);
- канальный (синоним — уровень звена данных 2);
- физический (уровень 1).

Верхним уровнем является прикладной, который состоит из прикладных объектов. Нижележащие уровни представляют услуги, посредством которых осуществляется взаимодействие между прикладными объектами.

Уровни 1...6 вместе с физическими средствами соединения для ВОС обеспечивают пошаговое расширение услуг связи. Граница между двумя уровнями обозначает ступень в этом расширении услуг.

*Прикладной уровень* обеспечивает доступ прикладных процессов к среде ВОС. Он выполняет функции «окна» между взаимодействующими прикладными процессами, которые используют среду ВОС для совместного решения задач.

Функции прикладного уровня разделяются на две группы: общие и специальные прикладные сервисные элементы. Первые

дают средства взаимодействия, используемые рядом различных приложений (например, средства организации связи между прикладными процессами). Вторые обеспечивают определенные потребности конкретных приложений (например, обмен файлами, передача заданий).

*Представительный уровень*

Назначением представительного уровня является представление данных, подлежащих передаче между прикладными объектами; представление структуры данных, на которую прикладные объекты ссылаются в процессе своего обмена, наряду с представлениями совокупности действий, которые могут быть выполнены над этой структурой данных.

Представительный уровень имеет дело с синтаксисом, то есть с представлением данных, а не с их семантикой, то есть их смыслом, известным только прикладным объектам. Представление данных в едином виде освобождает прикладные объекты от необходимости заботиться о проблеме «общего» представления информации, то есть обеспечивает для них независимость от синтаксиса. Эта независимость от синтаксиса достигается тем, что прикладные объекты могут использовать любой локальный синтаксис, а представительный уровень обеспечивает преобразование между этими синтаксисами и общим синтаксисом, необходимым для взаимодействия прикладных объектов. Это преобразование выполняется внутри открытых систем. Оно не видно для других открытых систем и поэтому не оказывает влияния на стандартизацию протоколов представления.

Функции представительного уровня сводятся к запросу на установление сеанса; передаче данных; согласованию и пересогласованию выбора синтаксиса; преобразованию синтаксиса, включая преобразование данных, форматирование и специальные преобразования (например, сжатие); запросу на прекращение сеанса.

Сущность второй и третьей функции заключается в следующем.

Существуют три варианта синтаксиса данных; синтаксис, используемый прикладным объектом-отправителем, прикладным объектом-получателем и используемый между представительными объектами (синтаксис передачи). Любые два или все три синтаксиса могут быть идентичными. Представительный уровень содержит функции, необходимые для выполнения преобразования между синтаксисом передачи и каждым из остальных двух синтаксисов по мере необходимости.



Для всей области ВОС нет единого заранее установленного синтаксиса передачи. Синтаксис передачи, который будет использоваться по представительному соединению, оговаривается между представительными объектами-корреспондентами. Таким образом, представительный объект должен знать синтаксис своего прикладного объекта и оговоренный синтаксис передачи, идентификатор которого используется в протоколах представительного уровня.

Согласование синтаксиса осуществляется посредством диалога между представительными объектами для определения формы, которую будут иметь данные в процессе обмена в функциональной среде ВОС. В процессе согласования определяется, какие преобразования необходимо выполнить (если такая необходимость имеется) и где они должны выполняться. Согласование может быть ограничено фазой инициирования либо выполняться в любое время в ходе сеанса.

*Сеансовый уровень*

Назначение сеансового уровня заключается в обеспечении сервиса, необходимого взаимодействующим представительным объектам для организации и синхронизации своего диалога и управления обменом данными. Для этого сеансовый уровень предоставляет услуги по установлению сеансового соединения между двумя представительными объектами и поддержания упорядоченного взаимодействия при обмене данными между ними. Для осуществления передачи данных между представительными объектами сеанс отображается на транспортное соединение и использует последнее. Сеанс существует до тех пор, пока он не будет расторгнут представительными или сеансовыми объектами. Во время существования сеанса его услуги поддерживают состояние диалога даже, несмотря на потерю данных, которая может произойти на транспортном уровне.

Функции сеансового уровня сводятся к установлению и расторжению сеансового соединения; обмену нормальными и срочными данными; управлению взаимодействием; синхронизации сеанса; восстановлению сеанса.

Функции сеансового уровня тесно связаны с его сервисом, поскольку «собственные» (т. е. не инициированные с верхнего уровня) действия на сеансовом уровне практически отсутствуют.

*Транспортный уровень* обеспечивает прозрачную передачу данных между сеансовыми объектами и освобождает их от функций, связанных с надежной и экономически эффективной передачей данных. Уровень оптимизирует использование имеющихся сетевых ресурсов, то есть предоставляет транспортный сервис при минимальной стоимости. Эта оптимизация достигается в пределах ограничений, накладываемых, с одной стороны, запросами всех одновременно действующих сеансовых объектов, и, с другой стороны, параметрами сетевого сервиса, используемого транспортным уровнем.

Все протоколы, определяемые на транспортном уровне, предназначены для межконцевого взаимодействия, где концы определяются как транспортные объекты-корреспонденты. Поэтому транспортный уровень ориентирован на оконечные открытые системы ВОС, а транспортные протоколы используются только между оконечными открытыми системами ВОС.

Поскольку сетевой уровень обеспечивает сетевые соединения между любыми двумя транспортными объектами, включая случай использования подсетей, соединенных последовательно, транспортный уровень освобождается от необходимости заниматься маршрутизацией и ретрансляцией.

Транспортные функции, используемые для обеспечения запрашиваемого качества сервиса, зависят от предоставляемого сетевого сервиса и сводятся к следующим: отображение транспортного адреса на сетевой адрес; мультиплексирование и расщепление транспортных соединений на сетевые соединения; установление и расторжение транспортных соединений; управление потоком на отдельных соединениях; обнаружение ошибок и контроль качества обслуживания; исправление ошибок; сегментирование, блокирование и сцепление; передача срочных транспортных блоков данных.



*Сетевой уровень* обеспечивает установление, поддержание и разъединение сетевых соединений между открытыми системами, содержащими взаимодействующие прикладные объекты, а также предоставляет функциональные и процедурные средства для обмена блоками данных между транспортными объектами по сетевым соединениям. Сетевой уровень обеспечивает транспортным объектам независимость от аспектов маршрутизации и ретрансляции, связанных с установлением и использованием данного сетевого соединения. Сюда относится случай, когда несколько подсетей используются последовательно или параллельно. Сетевой уровень также «скрывает» от транспортных объектов способы использования нижележащих ресурсов.

Сетевой уровень выполняет следующие функции: маршрутизацию и ретрансляцию; организацию сетевых соединений; мультиплексирование сетевых соединений на канальное соединение; сегментирование и блокирование; обнаружение и исправление ошибок; организацию последовательности; управление потоком; передачу срочных данных; возврат в исходное состояние.

Сетевые соединения могут иметь различную конфигурацию — как простую двухточечную, так и образованную сложными комбинациями подсетей с различными характеристиками. Для облегчения представления сетевые функции разделяются на подуровни. Такое разделение более подробно описано в гл. 3, посвященной сервису сетевого уровня.

*Канальный уровень* обеспечивает функциональные и процедурные средства для установления, поддержания и расторжения канальных соединений между сетевыми объектами и передачи блоков данных. Канальное соединение (канал передачи данных) строится на одном или нескольких физических соединениях.

Канальный уровень обнаруживает и, возможно, исправляет ошибки, которые могут возникнуть на физическом уровне. Кроме того, канальный уровень позволяет сетевому уровню управлять взаимными соединениями физических каналов передачи данных в физическом уровне.

На этом уровне выполняются следующие функции: установление и расторжение соединения; расщепление канального соединения на несколько физических; управление последовательностью; обнаружение и исправление ошибок; управление потоком; управление соединением физических каналов передачи данных.

*Физический уровень* обеспечивает механические, электрические, функциональные и процедурные средства активизации, поддержания и деактивизации физических соединений для передачи бит между канальными объектами. В физическое соединение могут входить промежуточные открытые системы, каждая из которых ретранслирует передачу бит внутри физического уровня. Объекты физического уровня соединены друг с другом посредством физических средств передачи.

Функции физического уровня сводятся к активизации и деактивизации физического соединения, а также передаче данных.

Любой уровень может быть описан теми функциями, которые в нем выполняются. В общем случае они включают в себя:

- выбор протокола;
- установление и расторжение соединений;
- мультиплексирование и расщепление соединений;
- передачу обычных данных;
- передачу срочных данных;
- управление потоком данных;
- сегментирование, блокирование и сцепление данных;
- организацию последовательности;
- защиту от ошибок;
- маршрутизацию;
- ретрансляцию;

Рассмотрим теперь эти функции более подробно.

*Выбор протокола.* Как известно, протоколом называют набор соглашений по организации связи в пределах одного уровня. Каждый уровень может использовать несколько протоколов. Но для организации некоторого N-соединения необходимо, чтобы N-объекты выбрали единый N-протокол (иначе они просто не поймут друг друга).



*Установление и расторжение соединения.* Для установления N-соединения необходимо выполнение двух условий:

- чтобы (N-1)-уровень предоставил (N-1)-соединение (то есть, услуги по передаче данных N-уровня);
- чтобы оба N-объекта были способны выполнить обмен данными по протоколу.

Для выполнения первого требования необходимо, чтобы (N-1)-уровень имел соединение, предоставленное (N-2)-уровнем и т.д. до физического.

Расторжение N-соединения в нормальных условиях инициируется одним из связанных с ним (N+1)-объектов. Расторжение N-соединения может также инициироваться одним из поддерживающих его N-объектов в результате возникновения сбоя в N- или нижележащем уровне.

Следует отметить, что N- и (N-1)-соединения могут быть независимыми. Возможны две ситуации:

- расторжение N-соединения не ведет к расторжению (N-1)-соединения;
- расторжение (N-1)-соединения не ведет к расторжению N-соединения.

В первом случае сохраненное (N-1)-соединение может быть использовано для установления нового N-соединения. Второй случай связан с возможностью восстановления N-соединения даже в том случае, когда (N-1)-соединение расторгнуто. Такое восстановление обычно связано с организацией нового (N-1)-соединения и передачей по нему данных, однозначно идентифицирующих сохраненное N-соединение.

*Мультиплексирование и расщепление соединений*. Между N- и (N-1)соединениями возможны следующие отношения (рисунок 3.6): одно к одному; несколько N-соединений используют одно (N-1)-соединение (мультиплексирование); одно соединение использует несколько (N-1)-соединений (расщепление).

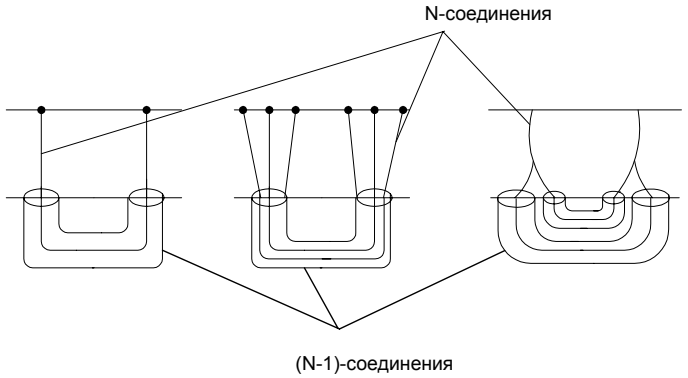



Рисунок 3.6 — Соотношение N и (N-1) соединений

При мультиплексировании необходимо осуществлять функции, связанные с идентификацией данных, относящихся к разным N-соединениям (поскольку эти данные поступают на N-уровень «вперемешку»), управлять и синхронизировать каждое N-соединение, чтобы исключить превышения нагрузочной способности, а также осуществлять планирование — предусматривать появление новых N-соединений, которые будут использовать сервис существующего (N-1)-соединения.

Расщепление связано с управлением (N-1)-соединениями, используемыми для одного N-соединения. Оно заключается в принятии решения о том, сколько (N-1)-соединений и с какими характеристиками необходимо запрашивать. Передаваемые по N-соединению данные дробятся и передаются по разным (N-1)-соединениям. Порядок их приема может отличаться от порядка передачи. Поэтому необходимы дополнительные функции контроля.

*Передача нормальных данных.* Взаимодействие N-объектов осуществляется с помощью обмена N-протокольными блоками данных (ПБД), которые содержат либо N-протокольную управляющую информацию, либо данные пользователя. Последними являются данные, генерируемые (N+1)-объектами. Эти данные передаются по N-соединению прозрачно, то есть без изменения их структуры.



*Передача срочных данных*. Срочные данные — это данные, которые обрабатываются с приоритетом по отношению к нормальным данным. Такие данные обычно используются для целей сигнализации, управления и т.д.

Поток срочных данных не зависит от состояния потока нормальных данных. Каждый из этих потоков образует своего рода подканал и управляется независимо. Поскольку предполагается, что поток срочных данных будет использоваться сравнительно редко и для малых количеств данных, для него могут быть применены упрощенные схемы управления потоком.

*Управление потоком данных*. Различают два типа управления потоком:

- протокольное, при котором регулируется скорость передачи N-ПБД между N-объектами;
- интерфейсное, при котором регулируется скорость передачи данных между (N+1)- и N-объектом.

*Сегментирование, блокирование и сцепление данных.* Протокольные блоки данных различных уровней обычно отличаются по размерам. Может оказаться, что размер (N+1)-ПБД больше максимального размера поля данных в N-ПБД. Тогда для передачи (N+1)-ПБД по N-соединению необходимо на N-уровне выполнить сегментацию, то есть разбиение (N+1)-ПБД на последовательные сегменты с длиной, соответствующей размеру поля данных N-ПБД. При этом, для сохранения целостности, в N-ПБД необходимо добавлять дополнительную информацию, обеспечивающую последующую сборку (N+1)-ПБД при приеме.

Блокирование — это объединение нескольких небольших (N+1)-ПБД в один N-ПБД.

Сцепление есть функция (N+1)-уровня, позволяющая объединить несколько (N+1)-ПБД в один блок. При этом, N-уровень воспринимает сцепленные ПБД как один блок — своего рода логическое блокирование.

*Организация последовательности*. Эта функция связана с тем, что (N-1)-услуги, предоставляемые (N-1)-уровнем, могут

не гарантировать доставку данных в том порядке, в каком они были представлены N-уровнем. Поэтому организация последовательности может потребовать дополнительной N-протокольной управляющей информации. Такой информацией могут быть, например, порядковые номера ПБД.

*Защита от ошибок*. Эта функция состоит из трех компонентов:

- подтверждения;
- обнаружения ошибок и уведомления о них;
- возврата в исходное состояние.

Подтверждение может использоваться N-объектами для достижения более высокой вероятности обнаружения потери N-ПБД, чем это обеспечивает (N-1)-уровень. Каждый N-ПБД, передаваемый между N-объектами, должен идентифицироваться единственным образом так, чтобы получатель мог информировать отправителя о его приеме.

Обнаружение ошибок и уведомление о них выполняют те же функции, что и подтверждение, но для N-протоколов по отношению к (N-1)-услуге. То есть служат для обеспечения N-протоколу более высокой вероятности обнаружения ошибок и искажений ПБД, чем это делает (N-1)-услуга.

Некоторые услуги требуют возврата в исходное состояние для восстановления после потери синхронизации между N-объектами. Функция возврата в исходное состояние устанавливает N-объекты в заранее определенное (по умолчанию) состояние с возможной потерей или дублированием данных.

*Маршрутизация*. Функция маршрутизации в N-уровне обеспечивает прохождение данных через цепочку N-объектов. Тот факт, что передача маршрутизируется промежуточными объектами, не известен ни нижним, ни верхним уровням. Объект, участвующий в выполнении функций маршрутизации, может иметь таблицу маршрутизации.

*Ретрансляция*. Не все открытые системы представляют собой начальный источник или конечный получатель данных. Если физические средства соединения не связывают их все непосредственно, то некоторые могут действовать как ретрансляционные. Функции и протоколы таких систем, поддерживающих пересылку данных, обеспечиваются на нижних уровнях (рис. 3.7).



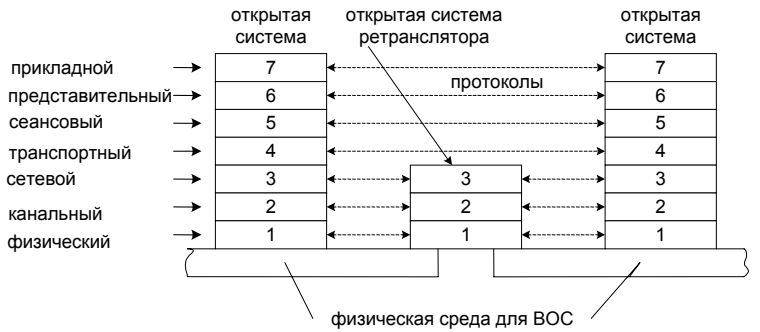



Рисунок 3.7 — Связь через открытую систему ретранслятора

## 3.3 Представление сервиса в модели ВОС

Как уже отмечалось, (N+1)-объекты могут связываться между собой только с помощью услуг, предоставляемых N-уровнем. Рассмотрим элементы сервиса N-уровня и понятия, необходимые для описания сервиса.

Объекты смежных уровней взаимодействуют друг с другом через общий интерфейс. Для локализации тех мест, в которых происходит взаимодействие, используется понятие сервисной точки доступа (СТД) N-уровня (N-СТД). Именно через N-СТД происходит предоставление сервиса N-уровнем и потребление сервиса (N+1)-уровнем.

Между (N+1)-объектами, N-объектами и N-СТД существуют определенные соотношения. Во-первых, объекты, имеющие общую СТД, находятся в одной системе. Во-вторых, (N+1)-объект может быть подключен к нескольким N-СТД, соединенным с одними и теми же несколькими N-объектами. Однако в каждый момент времени каждая N-СТД соединена только с одним N-объектом и только с одним (N+1)-объектом. Это жесткое ограничение обусловлено тем, что при обеспечении связи

объектов разных уровней СТД используется как идентификатор объекта соседнего уровня (рис. 3.8 ).

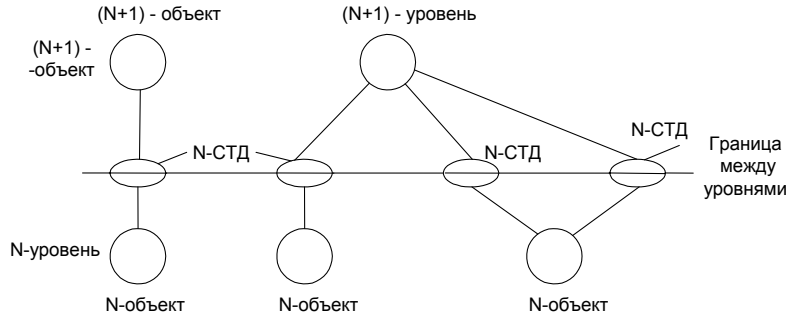



Рисунок 3.8 — Соотношение N-, (N+1)-объектов и N-СТД

Местоположение N-СТД определяется N-адресом. Если (N+1)-объект будет отключен от N-СТД, N-адрес не будет обеспечивать доступ к (N+1)-объекту. Если N-СТД переключается к другому (N+1)-объекту, то N-адрес идентифицирует новый (N+1)-объект.

В общем случае один адрес может идентифицировать несколько СТД. Однако для простоты будем считать, что между N-СТД и N-адресом существует отношение «один к одному». В иерархической системе ВОС доступ к объекту N-уровня обеспечивается использованием адресов на более низких уровнях.

Интерпретация соответствия между N-адресами, обслуживаемыми N-объектом, и (N+1)-адресами, используемыми для доступа к (N-1)-услугам, осуществляется с помощью функций отображения N-адресов.

Внутри уровня могут существовать два различных вида функций отображения N-адресов: иерархическое и отображение с помощью таблиц. Если N-адрес всегда отображается только в один (N-1)-адрес, может быть использовано иерархическое построение адресов. В этом случае N-адрес состоит из двух частей: (N-1)-адрес N-объекта, который поддерживает несколько N-



СТД; N-суффикс, который делает возможной идентификацию единственной N-СТД.

Если это предыдущее условие не выполняется, то есть либо N-адрес может быть отображен в несколько (N-1)-адресов, либо N- адрес не отображается постоянно в один и тот же (N-1)-адрес, то иерархическое построение адреса невозможно и функция отображения N-адреса должна использовать таблицы для преобразования N-адресов в (N-1)-адреса.

В пределах каждого уровня между парой СТД, расположенных в различных системах, может быть установлено несколько соединений. Например, когда каждая система в отдельности предоставляет более одного сервиса и необходимо различать точки доступа к ним. В этом случае используют идентификатор конечной точки соединения (КТС).

Точка связывает три элемента (рис. 3.8 ): (N+1)-объект, N-объект и N-соединение. Идентификатор КТС должен быть уникальным в пределах СТД. Его значение выбирается локальным образом. При доступе к сервису N-уровня идентификатор КТС указывает на установленное N-соединение, по которому предполагается передача данных.

Данные, передаваемые через интерфейс за одно взаимодействие, называются интерфейсным блоком данных. Очевидно, что в различных реализациях и на различных уровнях такой блок может иметь различный формат и длину. В связи с этим при рассмотрении сервиса используют понятие сервисного блока данных (СБД). Сервисный блок данных — это данные пользователя (в общем случае несколько интерфейсных блоков данных), идентичность которых при передаче по соединению сохраняется. Это означает, что границы, форма и содержание СБД на двух КТС идентичны и не зависят от того, какими порциями и как СБД передавался по соединению.

## 3.4 Правила описания сервиса

Вообще, понятие сервиса как набора возможностей (услуг) уровня довольно сложно формализовать. Это связано с тем, что процедуры и форматы передаваемых на границе уровней дан-

ных определяются особенностями реализации. Например, если уровни функционально разнесены по разным устройствам, интерфейс между уровнями может (и должен) содержать процедуры защиты от ошибок. Эти процедуры не нужны, если уровни расположены в одной операционной среде. Поэтому процедуры обмена на интерфейсе в общем случае не стандартизируются.

Однако для того, чтобы обеспечить независимость уровней, необходимо уметь описывать их характеристики. Тогда при проектировании протокола не надо будет увязывать его процедуры с механизмами передачи на более низких уровнях. Достаточно будет указать порядок использования процедур и параметров, описывающих внешнее поведение уровня.

Для пояснения того, каким образом можно задать внешнее поведение протокольного уровня, рассмотрим пример. Пусть два протокольных объекта А и Б устанавливают соединение друг с другом. Объект А передает протокольный блок REQ. Объект Б, получив блок REQ, отвечает блоком RES. При завершении обмена блоками соединение считается установленным.

С точки зрения вышестоящего пользователя, предоставляемый здесь сервис по установлению соединения заключается в том, что:

- пользователь-инициатор (на стороне А) может послать запрос на установление соединения;
- пользователь-рецептор может получить сообщение о том, что его запрашивают, и передать ответ;
- пользователь-инициатор может получить подтверждение от рецептора о его согласии установить соединение.

Обозначим сообщения пользователей следующим образом:

- CONNECT request — запрос установления соединения, выдаваемый пользователем А;
- CONNECT indication — уведомление пользователя Б о наличии запроса от пользователя А;
- CONNECT response — ответ пользователя Б;
- CONNECT confirmation — подтверждение установления соединения, выдаваемое пользователю А.

Перечисленные сообщения выбраны и определены так, чтобы они имели прямую связь с протокольными блоками. Разумеется, эти сообщения могли быть определены по-другому.



Сообщения не могут поступать в произвольном порядке. Например, CONNECT confirmation не может поступить раньше, чем будет выдан CONNECT request. Поэтому элементом внешнего (сервисного) описания протокольного уровня должны быть допустимые последовательности сообщений пользователей.

Наконец, необходимо задать соотношения сообщений на разных концах соединения. В этих соотношениях должна быть отражена семантика предоставляемого сервиса. Например, необходимо показать (применительно к нашему примеру), что сообщение CONNECT indication есть следствие сообщения CONNECT request. Такие соотношения можно задавать диаграммами последовательности и таблицами соответствия параметров (рис. 3.9).

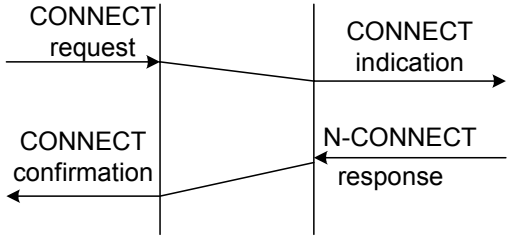



Рисунок 3.9 — Диаграмма последовательности пользовательских сообщений

Таблицы соответствия параметров устанавливают, должны ли быть равны параметры сообщений, имеющих последовательную зависимость. Пусть при установлении соединения пользователь А передает пользователю Б пароль, а пользователь Б — данные. Тогда предоставляемый сервис заключается в том, что значение пароля, указываемое в CONNECT indication, равно аналогичному значению в порождающем запросе CONNECT request, а поле данных в CONNECT confirmation равно полю данных в CONNECT response. Это означает, что уровень предоставляет сервис, при выполнении которого пароль и данные одного пользователя прозрачно (то есть без искажений и потерь) передаются другому пользователю.

Рассмотренные пользовательские сообщения получили в литературе и стандартах ВОС название сервисных примитивов. Это название отражает тот факт, что сервисные примитивы представляют первичные, неделимые элементы описания сервиса. Каждый сервисный примитив является именованной, то есть имеющей уникальное название, совокупностью параметров.

Использование примитивов не предполагает, что для реализации интерфейса достаточно только описания сервиса. Сервисные примитивы — это концептуальные понятия, облегчающие описание последовательности событий при доступе к сервису уровня. Тот же сервис может быть описан другим, эквивалентным набором примитивов. Кроме того, сервисные примитивы не используются при описании чисто локальных действий, не связанных с обменом данными между пользователями.

Сервис состоит из услуг. Так, показанный ранее пример — это сервис, состоящий из одной услуги по установлению соединения. Услуги могут быть обязательными и факультативными, а также подтвержденными и неподтвержденными. Обязательность услуги означает, что она должна предоставляться во всех реализациях. Факультативные услуги могут предоставляться или нет в зависимости от назначения реализации. Подтверждаемые услуги — это те, предоставление которых связано с обменом парой сервисных примитивов — примитивом запроса и примитивом подтверждения. Для некоторых неподтверждаемых услуг обмен сервисными примитивами отсутствует — здесь достаточно только передачи запроса от пользователя сервиса.

Изложенные понятия являются основой формализованного описания сервиса. Элементы такого описания в настоящее время стандартизованы и называются соглашениями по сервису. Основная цель этого стандарта — дать единое понимание сервиса, необходимое при использовании других стандартов, определяющих сервис конкретных уровней модели ВОС.

Согласно принятому стандарту, сервис уровня определяется через элементы абстрактной модели взаимодействия пользователей сервиса и поставщика сервиса. Эта модель включает в себя следующие понятия (рис. 3.10): пользователи N-сервиса, поставщик N-сервиса и сервисные примитивы. Последние разделяются на примитивы: запроса (request), индикации (indication), ответа (response) и подтверждения (confirmation).



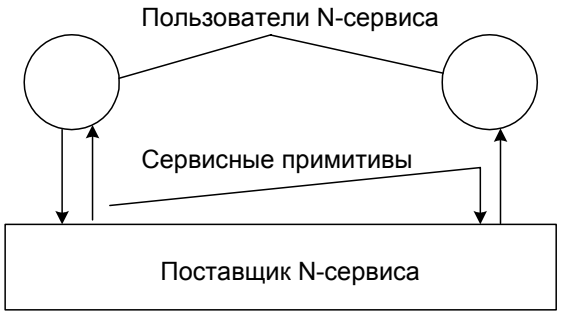



Рисунок 3.10 — Элементы модели сервиса

На практике список типов примитивов более велик, однако, указанных четырех типов вполне достаточно для описания сервиса уровней модели ВОС. Тип примитива связан с направлением его передачи. Примитивы запроса и ответа передаются от пользователя сервиса к поставщику, а примитивы индикации и подтверждения — в обратном направлении. Обозначение каждого сервисного примитива состоит из трех элементов: буквы (или букв), обозначающей уровень модели ВОС; имени примитива, отвечающего типу описываемой услуги; типа примитива.

Уровень обозначается буквами A (прикладной), P (представительный), S (сеансовый), T (транспортный), N (сетевой), DL (канальный) и PL (физический).

Имя примитива определяется типом услуги. Например, услуга по установлению соединения описывается примитивами с именем CONNECT, по сбросу — RESET, по передаче данных — DATA и т.д.

Таким образом, примитив P — CONNECT request есть примитив представительного сервиса, относится к услуге по установлению соединения и является запросом.

Стандартом также оговариваются формальные правила составления диаграмм последовательностей примитивов, примеры которых были приведены ранее. Каждая диаграмма представля-

ется тремя полями, разделенными двумя вертикальными линиями (рис. 3.11). Центральное поле представляет поставщика сервиса, а крайние поля — пользователей сервиса. Вертикальные линии изображают точки доступа к сервису и, кроме того, течение времени (сверху вниз).

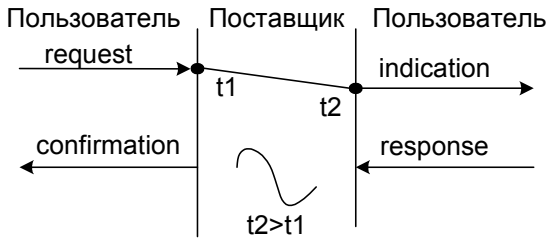



Рисунок 3.11 — Элементы стандартной диаграммы последовательности

Если между сервисными примитивами существует явная причинно-временная зависимость, они соединяются прямыми. Если же явной зависимости примитивов нет, то используется знак тильды.

Дальнейшее развитие абстрактной модели сервиса связано с введением в нее элементов поведения поставщика сервиса. Работа поставщика сервиса представляется операциями над двумя очередями, соединяющими сервисные точки доступа (рис. 3.12). Эти две очереди представляют одно соединение. Пользователь может помещать в очередь содержимое сервисных примитивов, то есть их параметры, и октеты данных. Некоторые параметры могут вставляться поставщиком (например, относящиеся к разъединению).

Концепция очереди имеет следующий смысл. В исходный момент очередь пуста, пользователь сервиса вставляет в очередь какой-либо элемент (например, запрос соединения). Помещение в очередь других элементов ограничено правилами их следования. На другом конце элементы извлекаются из очереди (обычно без изменения порядка их следования). Чтобы очередь не переполнилась, вводятся ограничения. Например, если добавление



в очередь элемента данных не позволит затем добавить в нее элемент разъединения, такое действие является запрещенным.

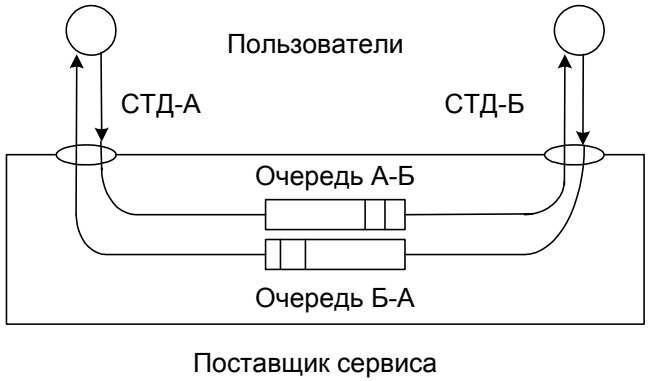



Рисунок 3.12 — Модель поставщика сервиса

Поставщику сервиса разрешается манипулировать некоторыми элементами очереди. Он может поменять местами два элемента, если последним является, например, элемент разъединения. Возможные переупорядочивания задаются таблицами предшествования, в которых указывается, какие элементы могут быть переставлены вперед.

Поставщик сервиса может также аннулировать элементы очереди. Например, операция сброса на сетевом соединении приводит к аннулированию всех имеющихся в очереди октетов данных.

Все рассмотренные правила задания сервиса относились к одному соединению. Именно поэтому не было необходимости различать сервисные примитивы, относящиеся к разным соединениям. В общем случае каждый примитив должен содержать идентификатор конечной точки соединения, к которому он относится. В дальнейшем при описании сервиса уровней будет считаться, что такой идентификатор содержится в каждом примитиве.

## 3.5 Взаимосвязь системы в режиме «без соединения»

Одним из основных понятий эталонной модели ВОС является соединение. В рамках модели соединение рассматривается как единственное средство взаимодействия объектов уровня. Вместе с тем ряд положений модели (концепция уровней, сервиса, протокола) не связаны с механизмом взаимодействия объектов. Они могут быть адекватно применимы и для режимов, когда соединение не устанавливается, а взаимодействие осуществляется с передачей отдельных независимых блоков данных. Такие режимы используются довольно широко в сетях передачи данных датаграммного типа, локальных сетях, при реализации некоторых типов прикладных процессов. Поэтому еще в 1985 г. был разработан первый проект приложения к эталонной модели ВОС, касающейся взаимосвязи открытых систем в режиме «без соединения».

Понятие передачи без соединения означает, что блок данных передается одним независимым действием без установления, поддержания и разрыва соединения. Чтобы лучше понять, что означает отказ от соединения, рассмотрим основные его свойства.

Соединение представляет собой логическую связь, устанавливаемую между двумя или более объектами для их взаимодействия. Объекты могут принадлежать одному либо соседним уровням (ассоциация пользователя сервиса и поставщика). Возможность установить связь обеспечивается нижележащим уровнем, предоставляющим сервис «с соединением». В процессе использования этого сервиса различают три различные фазы: установление соединения, передачу данных и разъединение. Каждая фаза, кроме процедурных отличий, имеет четко выраженные начало и конец. Характерной особенностью здесь является то, что:

- при установлении соединения производится согласование параметров соединения, режима передачи данных, использования необязательных процедур и т.д.;

- обеспечивается идентификация соединения, позволяющая при последующей передаче данных обойтись без ряда параметров (например, не использовать адрес);



- обеспечивается логическая связь передаваемых по соединению фрагментов данных, что позволяет эффективно сохранять последовательность и управлять потоком данных.

Режим «без соединения», в противоположность описанным свойствам соединения, не имеет четко выраженных фаз взаимодействия. Динамическое согласование параметров передаваемых данных также отсутствует, хотя необходима предварительная договоренность объектов (иначе объект «не поймет» принятые данные). Вся информация, требуемая для доставки данных — адреса, параметры качества сервиса, факультативные процедуры и т.д. — должны указываться при передаче каждого блока данных. Наконец, отсутствие логической связи между блоками данных позволяет передавать блоки по разным маршрутам, а также копировать блоки для широковещательной рассылки.

Таким образом, каждый из режимов имеет четко выраженные отличия. Сравнивать эти режимы на выявление лучшего не имеет смысла. Однако правомерно поставить вопрос об областях их применения.

Режим «с соединением» целесообразно использовать для тех применений, где взаимодействие имеет долговременный характер, конфигурация взаимодействующих объектов постоянна, а поток данных однороден (нет больших пауз).

Режим «без соединения» больше подходит там, где взаимодействие носит «точечный» характер, при котором объем передаваемых данных невелик, а интервалы между передачами значительны (относительно скорости передачи) — чаще всего для передачи служебной информации. Кроме того, легкость тиражирования данных и передачи их по разным маршрутам делают этот режим привлекательным в системах, к которым предъявляются повышенные требования устойчивости к отказам.

С точки зрения поставщика сервиса «без соединения», пользователь должен задать ему адреса объектов-корреспондентов и идентификатор используемого протокола, гарантировать факт наличия партнера и его готовности принять данные, а также указать качество предоставляемого сервиса.

С архитектурной точки зрения режим «без соединения» приводит к выделению отдельных N-объектов, связанных с выполнением функций протокола «без соединения». Поскольку каждый N-объект идентифицируется (N-1)-СТД (или несколькими (N-1)-СТД), возможны следующие случаи:

1. (N-1)-СТД поддерживает только сервис «без соединения».
2. (N-1)-СТД поддерживает только сервис «с соединением».
3. (N-1)-СТД поддерживает оба вида сервиса. Последнее возможно, когда N-протокол имеет средства различения двух режимов.

Документ не налагает архитектурных ограничений на комбинацию режимов. Так, N-объекты, поддерживающие режим «без соединения», могут использовать (N-1)-сервис типа «с соединениями». Возможно и обратное.

Передача данных с установлением соединения более надежна, но требует больше времени для передачи данных и вычислительных затрат от конечных узлов.

## 3.6 Физический уровень

Задача физического уровня — обеспечить соединение для передачи физических СБД (ФСБД). Для этого физический уровень формирует передаваемый сигнал, кодирует, декодирует и синхронизирует биты данных, контролирует состояние среды передачи и может также восстановить искаженные элементы принятого сигнала.

Сервис физического уровня образуется из следующих услуг: организации физических соединений; идентификации КТС; идентификации физических соединений; передачи ФСБД; организации последовательности; оповещения о возникновении отказа; обеспечения выбора качества сервиса.

Физическое соединение предоставляется пользователям, расположенным на канальном уровне. Конечные точки физического соединения находятся на границе с канальным уровнем (рис. 3.13).



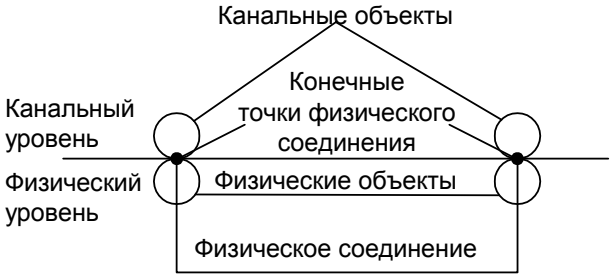



Рисунок 3.13 — Конечные точки физического соединения

Физическое соединение может иметь несколько КТС (рис. 3.14), если требуются параллельные соединения либо многоточечная конфигурация. В каждом случае физическое соединение идентифицируется системами, между которыми оно существует.

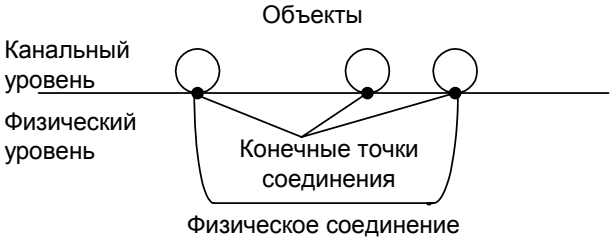



Рисунок 3.14 — Многоточечное физическое соединение

Физическое соединение представляет собой канал передачи данных в физической среде ВОС. Организация физического соединения связана с активацией объектов уровня. Поскольку физическая среда соединяет не все объекты физического уровня, некоторые объекты выполняют роль ретрансляторов, позволяющих «провести» физическое соединение до требуемой системы (рис. 3.15).

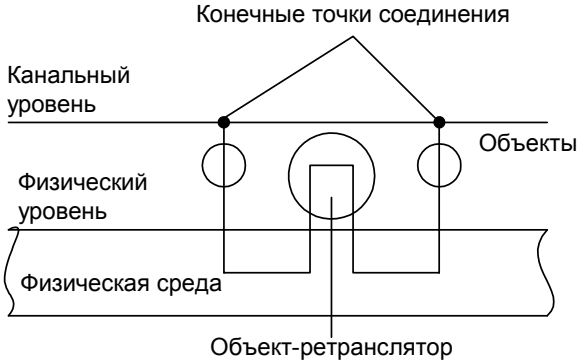



Рисунок 3.15 — Образование физического соединения через ретранслятор

Сервисным блоком данных физического уровня является 1 бит в случае последовательной передачи и N бит при параллельной.

Биты передаются дуплексным или полудуплексным способом и доставляются потребителю в том же порядке, в каком они поступали от источника. Если внутри физического уровня обнаруживается отказ (например, нарушение физической среды), то канальные объекты оповещаются об этом.

Качество физического сервиса зависит от физических каналов передачи данных, образующих соединение. Качество сервиса определяется следующими характеристиками: частотой появления ошибок, которые могут возникать в результате искажения, потери, вставки и других причин; доступностью сервиса; скоростью передачи; транзитной задержкой (задержкой передачи).

Физический уровень занимает в эталонной модели особое место, поскольку он не имеет поддержки нижележащих уровней, а опирается непосредственно на физическую среду. Развитие систем связи, передачи и телеобработки данных привело к возникновению большого числа стандартов на правила передачи сигналов через разнообразные физические среды.



## 3.7 Канальный уровень

Подавляющее большинство стандартов канального уровня было разработано до появления эталонной модели ВОС. Эти стандарты ориентировались на существующие системы телеобработки и передачи данных, а позднее — на особенности применяемых средств передачи и типов сетей. В настоящее время существует большое число разнообразных стандартов канального уровня, и работа над их развитием и расширением продолжается.

Исходя из характера представления передаваемой информации канальные протоколы подразделяются на знак-ориентированные (позначные) и бит-ориентированные. В первом случае наименьшей интерпретируемой единицей является знак (в определенном коде), а во втором — бит информации.

Первые стандарты от International Standard Organisation (ISO) для канального уровня появились в начале 70-х гг. и относились к процедурам синхронной позначной передачи (так называемый основной режим). Эти стандарты были основаны на ряде национальных и фирменных стандартов (в частности, на протоколе BSC фирмы IBM).

Позже были разработаны стандарты нового поколения канальных процедур. Эти стандарты получили название высокоуровневых процедур управления каналом данных и были ориентированы на синхронную побитовую передачу данных. По сравнению с процедурами основного режима высокоуровневые процедуры имели ряд преимуществ:

- отсутствие ориентации на определенный код;
- возможность работы на каналах различной конфигурации (то есть точка — точка и многоточечный); более эффективные способы управления потоком и исправления ошибок.

Дальнейшее развитие канальных стандартов происходит по двум направлениям — разработка многоканальных процедур и процедур для локальных сетей.

Многоканальные процедуры используются для повышения надежности передачи и пропускной способности соединения за счет параллельной работы по нескольким физическим соединениям. Каждое физическое соединение используется для организации подканала, а многоканальная процедура «собирает» эти

подканалы в одно соединение, предоставляемое объектам сетевого уровня.

Специфика протоколов локальных сетей заключается в том, что функции канального уровня здесь расширены за счет добавления процедур управления доступом к среде. Стандарты на канальный протокол локальных сетей регламентируют сервис двух типов: с соединением и без соединения.

### 3.7.1 Канальный сервис с соединением

Разработка стандарта на сервис канального уровня при наличии разнообразных протоколов связана с выделением наиболее общих услуг, типичных для протоколов этого уровня. В этом смысле стандартный сервис есть сервис идеализированного уровня, который не соответствует реальному сервису, предоставляемому отдельным протоколом. Стандарт на канальный сервис может, как превосходить набор услуг реального протокола, так и не учитывать услуг, предоставляемых специализированными протоколами.

С точки зрения пользователей, то есть сетевых объектов, сервис канального уровня позволяет обеспечить:

- независимость от используемых физических средств передачи. Пользователи освобождаются от всех проблем, связанных с конфигурацией физического соединения или его техническими и процедурными характеристиками. Например, пользователь не знает, какой способ передачи — дуплексный или полудуплексный — используется;

- прозрачную передачу данных. Это означает, что пользователь может передавать данные с любым содержанием, форматом или кодировкой. Канальный уровень не интерпретирует эти данные, то есть доставляет их прозрачно;

- надежный обмен данными. Большое число ситуаций, связанных с потерей, переупорядочиванием или искажением данных, обрабатывается без вмешательства пользователей. Тем самым повышается вероятность безошибочной передачи данных;

- выбор качества сервиса. Качество сервиса канального соединения связано с такими параметрами, как пропускная способность, транзитная задержка, уровень ошибок, надежность. Пользователям предоставляется возможность запросить и согласовать параметры качества сервиса;



- установление соединения по требуемому адресу. Если на канальном уровне используется многоточечная конфигурация, то есть конфигурация, когда достижимы пользователи с разными адресами, то пользователю дается возможность указать необходимый канальный адрес (то есть адрес канальной СТД).

Канальный сервис состоит из следующих услуг: по установлению канального соединения; по согласованию качества сервиса (КЧС); по передаче канальных СБД (КСБД); связанной с управлением потоком КСБД; по передаче отдельных срочных КСБД (факультативная услуга); по переводу звеньевого соединения в исходное состояние с уведомлением об этом пользователей; по расторжению соединения, сопровождаемому возможной потерей данных.

*Установление канального соединения*. Услуга описывается следующими сервисными примитивами с параметрами (рис. 3.16):

1. DL-CONNECT request: (вызываемый адрес (Адрес вызываемой СТД), вызывающий адрес (адрес вызывающей СТД), использование срочных данных, параметры КЧС, данные пользователя).

2. DL-CONNECT indication: (вызываемый адрес, вызывающий адрес, использование срочных данных, параметры КЧС, данные пользователя).

3. DL-CONNECT response: (адрес ответчика (адрес отвечающей СТД), использование срочных данных, параметры КЧС, данные пользователя).

4. DL-CONNECT confirmation: (адрес ответчика, использование срочных данных, параметры КЧС, данные пользователя).

При поступлении запросов DL-CONNECT request одновременно с двух сторон возможны следующие исходы: установлено одно соединение; установлены два соединения; не установлено ни одного соединения.

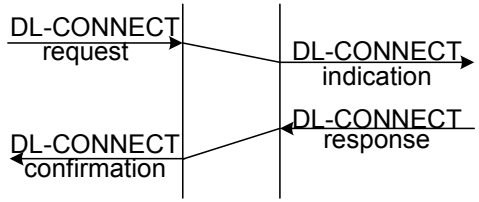



Рисунок 3.16 — Соотношение сервисных примитивов при установлении звеньевого соединения

Пользователи должны быть уведомлены об этих возможностях, поскольку они могут «подстроить» свое поведение для достижения необходимого результата (например, установить, что один партнер всегда является инициатором, а другой — рецептором).

Установление канального соединения связано с обменом параметрами сервисных примитивов. Соответствие параметров показано в таблице 3.1. Адреса в сервисных примитивах есть адреса канальных СТД. Использование адреса ответчика предусматривается для случаев, когда в многоточечной конфигурации возможна переадресация запроса соединения.

Таблица 3.1

| Параметр | Сервисный примитив | | | |
|---|---|---|---|---|
| | request | indication | response | confirmation |
| Вызываемый адрес | 0 | 0 | | |
| Вызывающий адрес | 0 | 0 | | |
| Адрес ответчика | | | 0 | 0 |
| Использование срочных данных | 0 | 0 | 0 | (=) |
| Параметры КЧС | 0 | 0 | 0 | (=) |
| Данные пользователя | П | (=) | П | (=) |

*0 – наличие параметра обязательно;*

*(=) – значение параметра должно быть равно аналогичному в порождающем сообщении.*



Использование срочных данных применяется тогда, когда предоставляется факультативная услуга по передаче срочных данных. Значения параметра — «использовать» и «не использовать».

При установлении соединения выполняется услуга по согласованию таких параметров КЧС, как пропускная способность и транзитная задержка для каждого направления передачи.

Для этих параметров согласовываются четыре значения: желаемое, крайнее приемлемое, обеспечиваемое и выбранное. Первые два значения определяются пользователем в примитиве DL-CONNECT request. Второе и третье значения вставляются канальным уровнем в примитив DL-CONNECT indication. Наконец, четвертое значение вставляется в примитивы DL-CONNECT response и DL-CONNECT confirmation.

Два параметра КЧС не согласовываются, но выбираются пользователем — это тип защиты данных и приоритет.

*Разъединение звеньевого соединения*. Услуга связана с обменом следующими сервисными примитивами:

1. DL-DISCONNECT request: (инициатор, причина, данные пользователя);

2. DL-DISCONNECT indication: (инициатор, причина, данные пользователя). Эти сервисные примитивы используются в тех случаях, когда разъединение инициировано:

- одним или обоими пользователями на уже установленном соединении (рис. 3.17,*а*);
- канальным уровнем на уже установленном соединении (рис. 3.17,*б*);
- пользователем при отказе установить соединение (3.17,*в*);
- канальным уровнем при невозможности установить соединение (3.17,*г*);
- пользователем, который перед этим выдал DL-CONNECT request (3.17,*д*).

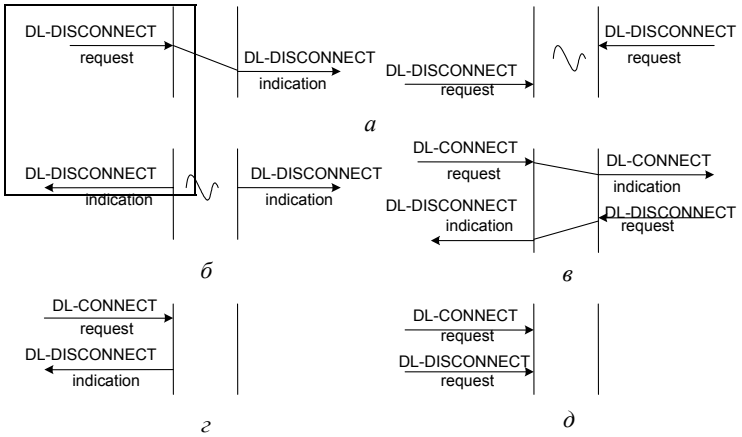



Рисунок 3.17 — Соотношение сервисных примитивов при разъединении звеньевого соединения

Использовать услугу по разъединению разрешается в любой момент. При разъединении могут быть аннулированы не доставленные данные. Используемые при разъединении параметры имеют следующий смысл:

- инициатор — указывает источник разъединения (пользователь или канальный уровень);

- причина — дает информацию о причине разъединения. Порядок использования этого поля приведен в таблице 3.2;

- данные пользователя — могут иметь ограничение на длину в 128 октетов.

*Передача данных.* При передаче обычных (не срочных) данных используются следующие сервисные примитивы:

DL-DATA request: (данные пользователя);

DL-DATA indication: (данные пользователя).

Данные пользователя в примитивах представляют собой КСБД. Услуга по передаче данных заключается в прозрачной передаче КСБД от одной СТД к другой с сохранением последовательности КСБД. Для каждой пары примитивов задана последовательность использования при выполнении услуги (рис. 3.18). Соотношение может быть нарушено ситуациями разъединения или возврата в исходное состояние.



Таблица 3.2

| Значение поля «причина» | Значение поля «инициатор» | |
|---|---|---|
| | Канальный уровень | Пользователь |
| Долговременное разъединение | X | |
| Кратковременное разъединение | X | |
| Адрес неизвестен | X | |
| Адрес недостижим | X | |
| Требуемое КЧС не может быть обеспечено | X | |
| Не определено | X | |
| Нормальное разъединение | | X |
| Разъединение из-за сбоя | | X |
| Отказ от установления соединения | | X |

*X — используется.*

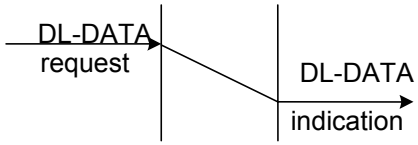



Рисунок 3.18 — Соотношение сервисных примитивов при передаче данных

Услуга по управлению потоком данных в явном виде (то есть через сервисные примитивы) не описывается. Неформально управление потоком выражается в том, что очередь, которая моделирует поведение поставщика в одном направлении, не переполняется (то есть возможность добавления данных в очередь зависит от частоты выборки данных из очереди принимающим пользователем). Описание факта добавления элемента в очередь связано с формализацией процедур интерфейсного управления потоком. Такие процедуры в настоящее время не входят в описание стандартного сервиса, поскольку явно не связаны с взаимностью объектов.

Факультативной услугой является передача срочных данных, описываемая следующими сервисными примитивами:

1. DL-EXPEDITED DATA request: (данные пользователя);
2. DL-EXPEDITED DATA indication: (данные пользователя).

Сама по себе эта услуга не отличается от услуги по передаче нормальных (не срочных) данных и имеет такое же соотношение сервисных примитивов.

По сравнению с потоком нормальных данных поток срочных данных имеет следующие свойства:

- срочные КСБД доставляют пользователю даже в том случае, когда пользователь приостанавливает прием нормальных КСБД;
- срочные КСБД могут обгонять нормальные КСБД.

Канальный уровень гарантирует, что срочный КСБД не будет передан пользователю позже любого нормального КСБД, перед которым срочный КСБД был выдан отправителем (рис. 3.19).

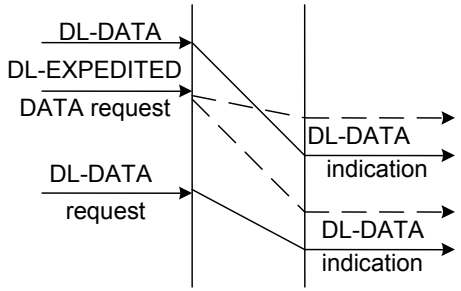



Рисунок 3.19 — Соотношение нормальных и срочных КСБД. Возможные отношения примитивов срочных данных показаны штриховой линией

*Перевод в исходное состояние*. Услуга по переводу в исходное состояние (синоним — услуга по сбросу) используется в двух случаях:

- когда пользователю необходимо очистить соединение от всех недоставленных данных и таким путем «сфазироваться» с объектом-партнером. Устранение недоставленных данных может также потребоваться при перегрузке соединения;
- когда канальный уровень обнаруживает неисправимую ошибку или невосстанавливаемую потерю данных.



Для выполнения услуги используются следующие сервисные примитивы:

1. DL-RESET request: (инициатор, причина);
2. DL-RESET indication: (инициатор, причина);
3. DL-RESET response;
4. DL-RESET confirmation.

Последовательность этих примитивов зависит от того, кто является инициатором их передачи. Различают, когда сброс инициирован одним из пользователей (рис. 3.20,*а*), обоими пользователями (рис. 3.20,*б*); канальным уровнем (рис. 3.20,*в*) или одновременно пользователем и канальным уровнем (рис. 3.20,*г*).

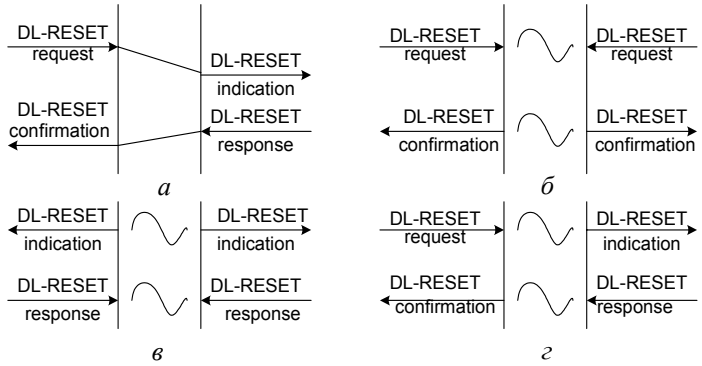



Рисунок 3.20 — Соотношение сервисных примитивов при переходе в исходное состояние

Параметры примитивов используются следующим образом:

- инициатор — указывает на источник сброса и может иметь два значения — «пользователь» или «канальный уровень»;
- причина — принимает значение «перегрузка соединения» или «ошибка», если инициатором является канальный уровень, и значение «рабочий сброс», если инициатором является пользователь.

Соответствие параметров примитивов такое же, как при разъединении соединения.

Для того чтобы услуга по переводу в исходное состояние действительно помогала «фазированию» пользователей, вводятся ограничения на соотношения нормальных КСБД, срочных КСБД и сервисных примитивов сброса (рис. 3.21):

- ни один КСБД, переданный до начала сброса, но не доставленный партнеру, не будет доставлен ни после начала, ни после завершения сброса (то есть такие КСБД будут аннулированы);
- все КСБД, переданные пользователем во время проведения сброса, будут аннулированы;
- ни один КСБД, переданный пользователем после завершения сброса, не будет доставлен партнеру раньше, чем завершится сброс у партнера.

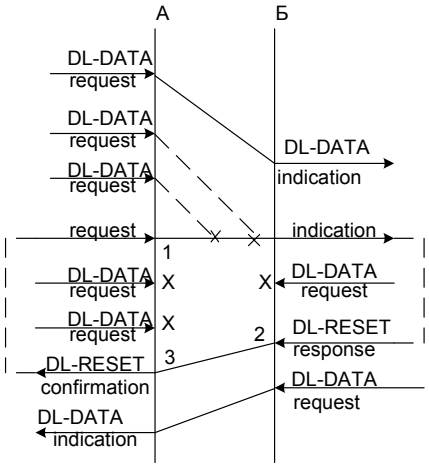



Рисунок 3.21 — Аннулирование КСБД при переводе в исходное состояние

Если в момент 1 выдается примитив DL-RESET request, то все данные, выданные системой А, но не доставленные партнеру, будут аннулированы. На стороне системы Б сброс начинается с момента получения примитива DL-RESET indication. Все



данные, выданные после этого момента, аннулируются. Завершение сброса на стороне Б наступает после выдачи примитива DL-RESET response (момент 2), а на стороне А — после приема DL-RESET confirmation (момент 3). Начиная с момента 2, сторона Б может передавать данные. Однако партнеру А эти данные не будут доставлены раньше момента 3, даже если эти данные являются срочными.

*Допустимые последовательности сервисных примитивов* описываются помеченным графом, вершины которого ассоциированы с состояниями конечной точки соединения, а ребра — с переданными сервисными примитивами (рис. 3.22).

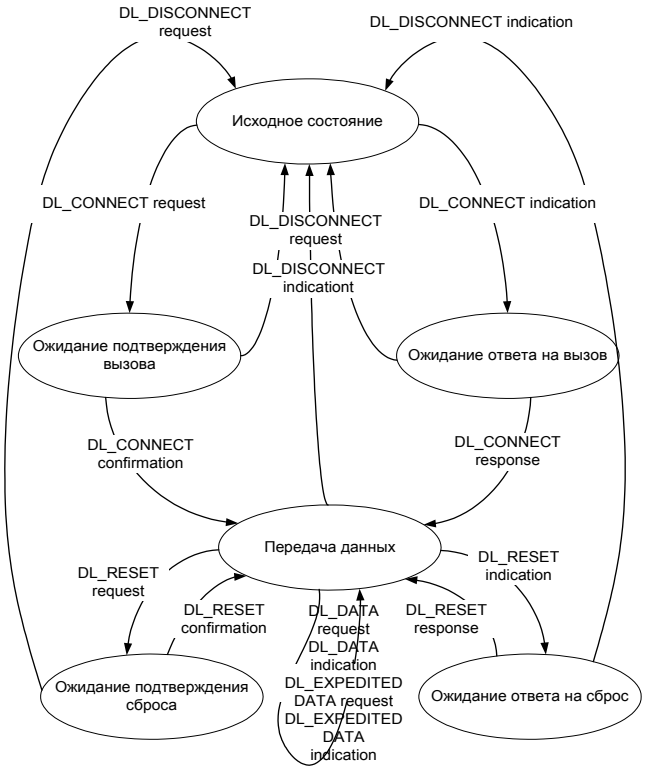



Рисунок 3.22 — Граф последовательности канальных сервисных примитивов

В исходном состоянии может быть передан DL-CONNECT request либо принят DL-CONNECT indication. Затем согласно соотношениям сервисных примитивов, описанным выше, при установлении соединения передается DL-CONNECT confirmation или DL-CONNECT response. Эти допустимые последовательности соответствуют переходам из исходного состояния в состояние передачи данных.

В состоянии передачи данных допускаются любые последовательности примитивов, связанных с передачей нормальных и срочных данных.

Переходы из состояния передачи данных в состояния ожидания подтверждения сброса, ожидания ответа на сброс и обратно описывают допустимые последовательности примитивов при сбросе. Эти переходы могут происходить в любой момент передачи данных.

Последовательность примитивов может быть завершена одним из примитивов разъединения DL-DISCONNECT request или DL-DISCONNECT indication, о чем свидетельствует наличие соответствующего перехода из каждого состояния в исходное.

Рассмотренный граф, задавая допустимые последовательности примитивов, не описывает взаимовлияния услуг. Например, здесь не отражается факт аннулирования данных при сбросе либо невыполнения услуги по установлению соединения при разъединении.

### 3.7.2 Параметры качества сервиса

Понятие «качество сервиса» (КЧС) относится к параметрам канального соединения в фазах его установления, разъединения и передачи данных. Параметры КЧС разделяются на три группы:

1) параметры, согласуемые во время установления соединения;

2) параметры, значения которых выбираются без согласования с партнером;

3) параметры, значения которых не выбираются, но сообщаются пользователям.



К первой группе принадлежат параметры пропускной способности и транзитной задержки, ко второй — параметры защиты соединения и приоритета, а третью группу образуют коэффициент необнаруженных ошибок и живучесть соединения. Знание параметров третьей группы необходимо пользователям для оценки надежности передачи и применения при необходимости дополнительных средств защиты от ошибок.

Согласование параметров первой группы не гарантирует сохранение их значений в течение всего времени существования соединения. При ухудшении параметров канальный уровень не прерывает соединения и не сообщает об этом пользователям. Пропускная способность соединения характеризует скорость ввода-вывода данных. Этот параметр основывается на заранее определенном размере КСБД и представляет собой среднее значение, рассчитанное по следующей методике (рис. 3.23):

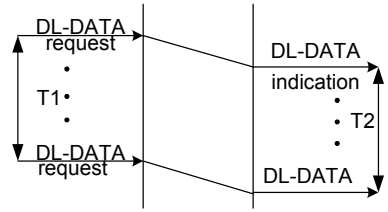



Рисунок 3.23 — Определение пропускной способности соединения

1) формируется последовательность примитивов DL-DATA request, каждый из которых содержит КСБД определенной длины;

2) фиксируется время T1, прошедшее между передачами первого и последнего примитивов DL-DATA request;

3) на другом конце соединения фиксируются только те примитивы DL-DATA indication, которые соответствуют безошибочной передаче данных с сохранением их последовательности;

4) определяется время T2, прошедшее между приемом первого и последнего примитива DL-DATA indication.

При подсчете *T1* и *T2* не учитываются задержки, вызванные пользователями при передаче примитивов DL-DATA request либо приеме примитивов DL-DATA indication. Значение параметра

пропускной способности равно меньшему из двух значений: а) числу бит в последовательности примитивов DL-DATA request, деленному на T1; б) числу бит в последовательности примитивов DL-DATA indication, деленному на T2.

Пропускная способность измеряется в битах в секунду для каждого направления передачи.

Транзитная задержка характеризует интервал времени между выдачей примитива DL-DATA request и появлением примитива DL-DATA indication. Этот параметр также основан на заранее определенном размере КСБД и представляет собой среднее значение, рассчитываемое для каждого направления передачи. Параметр транзитной задержки применим к отдельному КСБД. Он может быть использован для «предсказания» времени доставки КСБД при условии, что пользователь принимает КСБД без задержек, то есть не приостанавливая поток КСБД.

Защита соединения характеризует те меры, которые должен предпринять канальный уровень для защиты от несанкционированного доступа к передаваемым КСБД. Этот параметр характеризуется не количественно, а качественно. Пользователю могут быть предоставлены следующие возможности: отсутствие защиты; защита от пассивного подслушивания; защита от любой модификации данных (включая размножение, добавление или изъятие); защита от любого доступа к данным. Эти возможности не обязательно должны предусматриваться при всех реализациях, поскольку каждый из способов защиты обычно обеспечивается специализированным набором средств.

Приоритет соединения не следует путать с распространенной трактовкой приоритета как порядка обработки данных. Последний косвенно задается параметрами транзитной задержки. Приоритет соединения связан с относительной важностью соединения и характеризует порядок ухудшения КЧС на соединениях и разъединения установленных соединений для освобождения ресурсов. Другими словами, соединение с меньшим приоритетом характеризуется большей вероятностью ухудшения сервиса вплоть до разъединения.

Коэффициент (уровень) необнаруженных ошибок характеризует долю ошибок, которые канальный уровень не смог устранить или зафиксировать. Этот коэффициент определяется отношением общего числа искаженных, утраченных или дублированных КСБД ко всем переданным КСБД. В число утраченных входят также КСБД, доставленные по неверным адресам.



Живучесть (надежность) соединения характеризует его устойчивость в течение определенного периода времени. Параметр живучести определяет вероятность того, что канальный уровень инициирует разъединение соединения либо его сброс в течение установленного периода времени, например 1с.

### 3.7.3 Канальный сервис без соединения

Сервис типа «без соединения» связан с передачей отдельных независимых КСБД. Для случая успешной передачи этот сервис описывается двумя примитивами (рис. 3.24):

1. DL-UNITDATA request: (вызываемый адрес, вызывающий адрес, параметры КЧС, данные пользователя);
2. DL-UNITDATA indication: (вызываемый адрес, вызывающий адрес, параметры КЧС, данные пользователя).

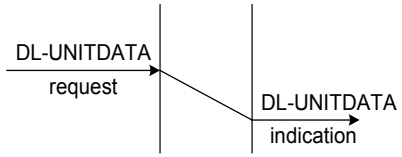



Рисунок 3.24 — Соотношения примитивов сервиса без соединения

Адресные параметры имеют те же значения, что и в примитивах сервиса с соединением.

Использование параметров КЧС связано со следующими правилами:

- в примитиве DL-UNITDATA request разрешается указывать любое значение параметров КЧС из диапазона обеспечиваемых;
- в примитиве DL-UNITDATA indication значение параметров КЧС не может быть лучше, чем запрошенное качество сервиса.

К параметрам КЧС относятся транзитная задержка, защита и коэффициент необнаруженных ошибок. Каждый из этих параметров передается в примитивах сервиса без соединения и означает:

- транзитная задержка — средний интервал времени между выдачей DL-UNITDATA request и получением соответствующего DL-UNITDATA indication;

- защита — способ защиты от несанкционированного доступа к данным в рамках тех же возможностей, что и для сервиса с соединением;

- коэффициент необнаруженных ошибок — имеет то же значение, что и для сервиса с соединением.

Параметр «данные пользователя» позволяет размещать КСБД любого размера в пределах установленного локально предела. Этот предел связан с размерами полей данных ПБД канального уровня.

Для более эффективного взаимодействия канального уровня и пользователей предложены факультативные средства, связанные с обменом дополнительной информацией на интерфейсе. Эти средства позволяют пользователю запросить данные о параметрах КЧС и получить некоторую информацию о состоянии передаваемых КСБД.

Запрос передается в сервисном примитиве:

DL-FACILITY request: (вызываемый адрес, требуемые средства).

При получении запроса канальный уровень сообщает параметры КЧС, ожидаемые при передаче КСБД по данному адресу. Кроме того, сообщается дополнительный параметр — вероятность сохранения последовательности. Этот параметр характеризуется отношением числа передач с сохранением последовательности КСБД к общему числу передач. Высокая вероятность сохранения последовательности может быть обеспечена при использовании надежных средств передачи данных. Для пользователя это дает возможность применять простые протоколы, то есть использовать более дешевые средства.

Информация о параметрах КЧС передается пользователю в примитиве:

DL-FACILITY indication: (вызываемый адрес, параметры КЧС).

Кроме того, пользователь может запросить специальную услугу, связанную с сообщением о состоянии передаваемых



КСБД. В случае если канальный уровень не способен передать КСБД (например, из-за перегрузки), он сообщает об этом примитивам:

DL-FACILITY report: (вызываемый адрес, причина отказа). В этом же примитиве может быть указано, что канальный уровень не может обеспечить те параметры КЧС, которые запрашиваются пользователем.

## 3.8 Сетевой уровень

В настоящее время уже существует большое число стандартов, относящихся к сетевому уровню. Однако интенсивно разрабатываются новые стандарты и дополнения к уже имеющимся. Это можно объяснить тем, что в стандартизации сетевого уровня заинтересовано значительное число организаций, представляющих производителей сетевых средств, пользователей этих средств и администрации сетей передачи данных. Сетевой уровень является границей между сетезависимыми и сетенезависимыми функциями, поскольку более высокие уровни обеспечивают взаимодействие типа «из конца в конец».

Для того чтобы уточнить место сетевых стандартов в эталонной модели ВОС, в МОС была разработана архитектура сетевого уровня. Рассмотрим основные положения этого стандарта, необходимые для понимания того, на чем базируется сервис сетевого уровня.

Особенность сетевого уровня состоит в том, что взаимодействие объектов абонентских систем может осуществляться через последовательность промежуточных систем, обеспечивающих функции коммутации (маршрутизации). Промежуточные системы образуют подсеть — совокупность технических средств и физической среды, представляющую собой автономное целое и используемую для взаимосвязи абонентских систем с целью обмена данными.

Реальная подсеть может быть реализована таким образом, что она полностью поддерживает стандартный сервис сетевого уровня. Однако большинство существующих и создаваемых подсетей предоставляет услуги, отличные от стандартных. Поэтому стандарт, оговаривая внутреннюю организацию сетевого

уровня, предусматривает, что услуги подсети могут быть идентичны стандартному сетевому сервису, полностью отличны от стандартного набора услуг или же отличны от стандартных в одних применениях, но идентичны в других. Если подсеть обеспечивает услуги второго и третьего типов, то на сетевом уровне для обеспечения стандартного сервиса необходимо выполнить дополнительные функции. Эти функции могут быть реализованы как в абонентских, так и в промежуточных системах.

Введение дополнительных функций взаимодействия приводит к появлению новых протоколов. В стандарте определены три типа таких протоколов:

1) независимые от подсетей протоколы конвергенции (НПП) ;

2) зависимые от подсетей протоколы конвергенции (ЗПП);

3) протоколы доступа к подсети (ПДП).

Под конвергенцией в данном контексте понимается сведение разнообразных услуг подсетей к единому стандартному набору.

В соответствии с тремя типами протоколов архитектура сетевого уровня может быть представлена в виде трех подуровней (рис. 3.25) : 3а — доступ к подсети, 3б — преобразования и протокол ориентированные и 3в — преобразования и протокол, не зависящие от подсети. Функциональность подуровней 3а и 3б у систем А и Б может быть различной (рис. 3.25). Промежуточная система здесь изображена для того, чтобы показать место функций маршрутизации в архитектуре уровня.

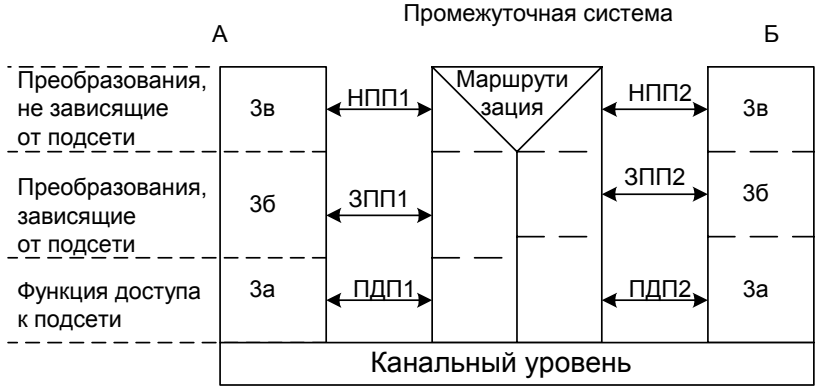



Рисунок 3.25 — Архитектура сетевого уровня



На подуровне 3а выполняются все функции, связанные с доступом к подсети передачи данных. Протокол подуровня 3а жестко ориентирован на тип подсети. Операции такого протокола вносят свой вклад в обеспечение услуг, характерных для соответствующей подсети, эти услуги могут совпадать или не совпадать с услугами сетевого уровня. Способ, посредством которого протокол вносит свой вклад в конструирование услуг сетевого уровня, зависит от соответствующих услуг подсети и способа использования этих услуг в конкретной конфигурации.

Функции подуровня 3б зависят от того, насколько сильно сервис подсети (или, точнее, сервис подуровня 3а) отличается от стандартного. Например, если сервис подуровня таков, что услуги по установлению и разъединению обеспечиваются полностью, а услуги по передаче данных не позволяют передавать данные неограниченной длины (как того требует стандартный сервис), то на подуровне 36 необходимо выполнить сборку/разборку сервисных блоков данных с использованием метки конца.

Функциональность уровня 3в зависит от того, какой сервис предоставляется нижележащим подуровнем. Для выбора необходимых услуг используются следующие критерии:

- типы подсетей и комбинации подсетей, которые предполагается охватить данным протоколом;

- технические и экономические преимущества и недостатки одного набора нижерасположенных функциональных возможностей по сравнению с другими;

- степень сложности тех протокольных механизмов в самом НПП, использование которых оправдано.

Для иллюстрации особенностей подуровня представим, что реальная подсеть, обеспечивающая взаимодействие абонентских систем, состоит из нескольких подсетей с отличными от стандартных услугами. В этом случае взаимодействие осуществляется через несколько подсетей, в каждой из которых используются различные идеологии, и имеется свой набор услуг. Если в каждой из подсетей отсутствует выравнивание сервиса подсети до стандартного (это может быть вызвано историческими или экономическими факторами), необходимы добавочные сетевые функции, которые бы не были ориентированы ни на одну подсеть. Именно такие функции выполняются на подуровне 3в.

### 3.8.1 Сетевой сервис с соединением

Сетевой сервис с соединением предоставляет пользователю следующие возможности:

- средства для установления сетевого соединения с другими пользователями для обмена сетевыми сервисными блоками данных (ССБД). Между парой пользователей может действовать несколько соединений;
- средства для согласования качества сервиса между двумя пользователями и поставщиком сервиса;
- средства для передачи ССБД по соединению. Передача ССБД, состоящих из целого числа октетов, является «прозрачной» в том смысле, что границы и содержание ССБД сохраняются неизменными и нет никаких ограничений на содержимое, формат или кодирование информации;
- средства управления потоком, с помощью которых принимающий пользователь может управлять скоростью передачи ССБД, посылаемых пользователем-отправителем;
- в некоторых случаях средство передачи отдельных срочных ССБД. Срочные ССБД ограничены по длине, и их передача подчинена отдельному (по сравнению с обычными данными) управлению потоком;
- средства использования услуги сброса, с помощью которой соединение может быть возвращено в исходное состояние, а действия обоих пользователей синхронизированы;
- в некоторых случаях средства подтверждения приема данных пользователем;
- безусловное разъединение соединения любым из пользователей или поставщиком.

Из перечисленных услуг подтверждение приема и передача срочных данных являются факультативными, все остальные — обязательными. Описание сетевого сервиса основано на использовании сервисных примитивов, приведенных в таблице 3.3.

Фазы установления соединения и разъединения сетевого уровня используют те же примитивы, что и канальный уровень, и здесь мы их приводить не будем. Рассмотрим только добавившуюся факультативную услугу по передаче данных с подтверждением приема.



Таблица 3.3

| Фаза соединения | Услуга | Примитив | Параметры |
|---|---|---|---|
| Установление | Установление соединения | N-CONNECT request<br>N-CONNECT indication | Вызываемый адрес, вызывающий адрес, выбор подтверждения приема, выбор использования срочных данных, параметры |
| | | N-CONNECT respons<br>N-CONNECT information | Адрес ответчика, выбор подтверждения приема, выбор использования срочных данных, параметры |
| Передача данных | Передача обычных данных | N-DATA request<br>N-DATA indication | Данные пользователя, запрос подтверждения |
| | Подтверждение приема* | N-DATA ACKNOWLEDGE request<br>N-DATA ACKNOWLEDGE indication | |
| | Подтверждение срочных данных* | N-EXPEDITED DATA request<br>N-EXPEDITED DATA indication | Данные пользователя |
| | Сброс | N-RESET request<br>N-RESET indication | Источник, причина |
| | | N-RESET respons<br>N-RESET confirmation | |
| Разъединение | Разъединение соединения | N-DISCCONNECT request<br>N-DISCCONNECT indication | Источник, причина, данные пользователя, адрес ответчика |

* - факультативная услуга

*Передача данных и подтверждение приема.* Услуга по передаче данных обеспечивает обмен ССБД по соединению. Сетевой уровень сохраняет как последовательность, так и границы ССБД и не лимитирует его длину. Особенностью примитивов этой услуги является наличие параметра «запас подтверждения». Этот параметр передается сетевым уровнем без изменений. Пользователь, установив значение параметра, может запросить подтверждения приема ССБД. Принимающий пользователь, получив примитив N-DATA indication с запросом подтверждения, выдает примитив N-DATA ACKNOWLEDGE request, который означает прием ССБД, содержащегося в примитиве индикации (рис. 3.26).

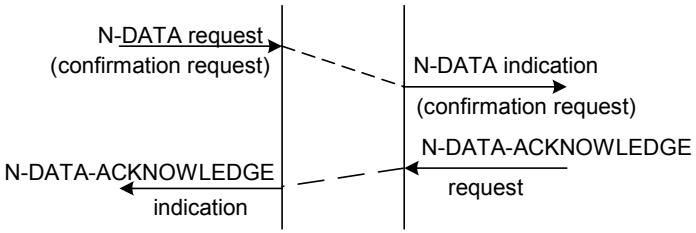



Рисунок 3.26 — Использование услуги подтверждения приема

Примитивы подтверждения приема данных передаются в потоке аналогичных примитивов так, чтобы пользователь мог соотнести принятое подтверждение с переданными ССБД простым подсчетом подтверждений.

На примитивы подтверждения приема не влияют процедуры управления потоком данных, в том числе управления на интерфейсах с пользователями.

*Передача срочных данных*. Услуга по передаче срочных данных дает дополнительные средства обмена ССБД ограниченной длины. Особенностью услуги является независимость от процедур управления потоком нормальными (не срочными) данными. Услуга обеспечивает такое же соотношение срочных и нормальных данных, как и аналогичная услуга на канальном уровне. Максимальный размер передаваемого срочного ССБД - 32 октета.



*Возвращение в исходное состояние*. Возвращение в исходное состояние (сброс) используется как пользователями для взаимной синхронизации, так и сетевым уровнем для сообщения о потере данных, если потеря не связана с разъединением соединения.

Особенность сброса заключается в том, что на сетевом уровне он приводит к уничтожению всех недоставленных данных, срочных данных и подтверждений приема. Тем самым может быть устранено переполнение соединения.

Соотношение примитивов данных и сброса такое же, как и для аналогичной канальной услуги. Однако соотношения примитивов сброса друг с другом имеют одну особенность — примитив N-RESET confirmation может выдаваться инициатору сброса до поступления от партнера примитива N-RESET response, то есть он может не иметь прямой зависимости. Это связано с тем, что в подсети передачи данных сброс осуществляется по участкам «узел-узел», при этом подтверждение сброса на одном участке имеет локальный характер и не зависит от подтверждения сброса на другом участке.

Также как и для канального уровня, мы можем задать последовательности сетевых сервисных примитивов с помощью графа или таблицы. Для сетевого уровня этот граф точно такого же вида, что и для канального. Единственное отличие: в состоянии передачи данных добавляется петля примитивов N-DATA-ACKNOWLEDGE.

Исходное состояние является начальным и конечным для любой последовательности примитивов. Те примитивы, которые отсутствуют в переходах из любого описанного состояния, являются запрещенными в этом состоянии. Например, запрещенными являются примитивы данных, срочных данных и запроса подтверждения в состояниях сброса.

Любая последовательность примитивов может быть завершена примитивами разъединения. Разъединение является услугой, выполняемой в любое время, о чем свидетельствует наличие переходов, связанных с разъединением, из любого состояния.

Последовательность не может состоять из одних примитивов разъединения. Это связано с тем, что исходное состояние равносильно отсутствию сетевого соединения. При нахождении

в исходном состоянии передача или прием примитивов разъединения невозможны, а выход из этого состояния всегда связан с передачей примитивов соединения.

### 3.8.2 Параметры качества сервиса

Качество сервиса сетевого уровня определяется качеством отдельных услуг нижележащих подуровней. В отличие от сервиса канального уровня, параметры КЧС которого в основном связаны с фазой передачи данных, перечень параметров КЧС сетевого сервиса дополнен характеристиками фаз установления и разъединения соединения. Этими характеристиками являются: задержка установления и вероятность неустановления для фазы установления соединения; задержка разъединения и вероятность неразъединения для фазы разъединения соединения.

В фазе передачи данных используются следующие параметры КЧС: пропускная способность, транзитная задержка, коэффициент необнаруженных ошибок, живучесть и вероятность отказа.

Большинство параметров КЧС фазы передачи данных определяется так же, как и в сервисе канального уровня. Исключением является параметр «вероятность отказа». Еще одним специфическим параметром является наибольшая приемлемая стоимость соединения.

Перечисленные дополнительные параметры отражают специфику сетевого уровня и позволяют более точно «настраивать» протоколы транспортного уровня на соединения различного качества. Эти параметры описываются следующим образом.

*Параметры фазы установления соединения*. Параметр «задержка установления» связан с оценкой наибольшей приемлемой задержки между выдачей примитива N-CONNECT request и появлением соответствующего примитива N-CONNECT confirmation. При расчете задержки учитывается также время, затрачиваемое партнером на выдачу примитива N-CONNECT response в ответ на примитив N-CONNECT indication.

При установлении соединения возможны отказы со стороны поставщика. Отношение числа отказов к общему числу попыток установления соединения характеризует вероятность неустановления. Под отказом здесь понимается невозможность установить соединение в пределах определенного периода времени из-за ошибок и сбоев на сетевом уровне, таких как неверное направление вызова, потеря вызова или его искажение.



Те случаи, когда установление соединения заканчивается неудачно по вине пользователя, не учитываются.

*Параметры фазы разъединения*. Параметр «задержка разъединения» характеризует наибольшую приемлемую задержку между примитивом N-DISCONNECT request и завершением разъединения сетевого соединения. Последнее означает, что пользователь может начать устанавливать новое соединение взамен разъединенного. Существенно, что задержка разъединения не связана с выдачей примитива N-DISCON-NECT indication у партнера и определяется только для стороны, инициирующей разъединение. Данное определение задержки разъединения предполагает, что пользователь, выдавший N-DISCONNECT request, получает в ответ локальное сообщение о выполнении разъединения.

Если это сообщение не поступает в течение некоторого тайм-аута, разъединение считается неудачным. Отношение числа неудачных разъединении к общему числу попыток характеризует вероятность неразъединения.

*Фаза передачи данных*. Рассмотренный ранее параметр живучести характеризовал вероятность того, что передача данных будет прервана процедурами разъединения или сброса. Дополнительной характеристикой надежности передачи данных является вероятность отказа.

Под отказом понимается ситуация, в которой пропускная способность соединения падает ниже минимально приемлемого уровня, а транзитная задержка и коэффициент необнаруженных ошибок становятся больше допустимых. Отношение числа отказов к общему числу ситуаций нормального функционирования характеризует вероятность отказа. Для определения того, что есть ситуация нормального функционирования, можно рассматривать ситуацию, когда требуемые параметры КЧС остаются

в заданных пределах в течение определенного периода времени. Таким периодом может быть, например, время существования сетевого соединения.

Специфическим параметром КЧС сетевого сервиса является наибольшая приемлемая для пользователя стоимость соединения. Порядок определения стоимости, градации этого параметра и способы его обеспечения требуют дальнейших исследований. Заметим, что этот параметр не передается партнеру и имеет, таким образом, влияние только на локальные средства.

### 3.8.3 Сетевой сервис без соединения

Передача ССБД в режиме «без соединения» означает, что каждый блок данных передается во время единичной операции взаимодействия с поставщиком сервиса. При этом между блоками данных отсутствуют какие-либо зависимости (связи). Последовательность блоков данных, передаваемых один за другим в один и тот же пункт назначения, не обязательно будет доставлена в том же порядке. Более того, не требуется, чтобы поставщик сервиса сообщал о недостатке данных или восстанавливал потерянные данные.

Поскольку сетевой сервис без соединения может обеспечиваться разными средствами нижележащих подуровней, в частности, средствами типа «с соединением», характеристики сервиса могут варьироваться в значительных пределах. В общем случае градации сервиса по отношению к передаваемым ССБД могут быть следующими:

- блоки могут быть аннулированы поставщиком, но не раньше установленного тайм-аута;
- блоки всегда аннулируются поставщиком после установленного тайм-аута;
- блоки могут быть аннулированы только при переполнении поставщика сервиса;
- блоки не аннулируются ни при каких условиях;
- последовательность блоков при передаче не меняется;
- блоки не дублируются.



Пользователь должен быть извещен о том, какая градация сервиса ему предоставляется. Эта информация поможет при выборе транспортного протокола.

Сервис без соединения так же, как и для канального уровня, описывается двумя примитивами:

1. N-UNITDATA request: (вызывающий адрес, вызываемый адрес, параметры КЧС, данные пользователя);
2. N-UNITDATA indication: (вызываемый адрес, вызывающий адрес, параметры КЧС, данные пользователя).

## 3.9 Сервис транспортного уровня

В соответствии с эталонной моделью ВОС транспортный уровень выполняет все необходимые процедуры для обеспечения надежной и эффективной передачи данных из конца в конец от одного пользователя (сеансового объекта) до другого. Предоставляемый сервис включает транспортный сервис с соединением и без соединения.

Транспортный уровень скрывает от пользователей (инкапсулирует) особенности используемой сети передачи данных (точнее, сетевого сервиса). Тип предоставляемого сервиса может быть не связан с типом потребляемого сервиса. Так, транспортный сервис с соединением может быть предоставлен над сетевым сервисом любого типа — как с соединением, так и без соединения.

Разнотипность применяемых на практике сетей передачи данных привела к появлению ряда стандартных транспортных протоколов и приложений к ним с различными наборами функций.

Транспортный сервис с соединением обеспечивается протоколом типа «с соединением» и приложениями к нему. Этот протокол фактически содержит пять различных протоколов, именуемых классами и ориентированных на разный сетевой сервис. Определяются три типа такого сервиса:

А — с приемлемым для пользователей уровнем необнаруживаемых ошибок и приемлемой частотой сообщений об обнаруженных ошибках;

В — с приемлемым уровнем необнаруживаемых ошибок, но приемлемой частотой сообщений об обнаруженных ошибках;

С — с неприемлемым уровнем необнаруживаемых ошибок и неприемлемой частотой сообщений об обнаруженных ошибках.

Каждый класс транспортного протокола имеет различную функциональность (рис. 3.27). Это связано с необходимостью обеспечивать, с одной стороны, надежный транспортный сервис над сетевым сервисом с ухудшенными характеристиками, а с другой стороны, устранять избыточные транспортные функции при использовании качественного сетевого сервиса.

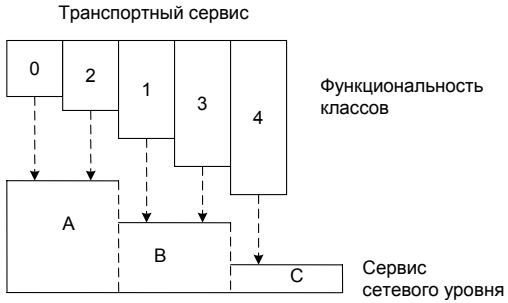



Рисунок 3.27 — Соотношение классов транспортного протокола и типов сетевого сервиса

На рисунке условно показано, что чем больше функциональность класса, тем хуже может быть качество используемого сетевого сервиса при сохранении неизменным транспортного сервиса. Наличие классов 0, 2 для сервиса типа А и классов 1,3 для сервиса типа В связано с оптимизацией использования сетевых ресурсов, поскольку классы 2 и 3 отличаются от классов 0 и 1 процедурами мультиплексирования транспортных соединений в сетевые. Функция мультиплексирования позволяет более рационально использовать сетевые ресурсы, а также снизить затраты на используемые сетевые соединения (особенно в тех случаях, когда стоимость установления нового соединения велика по сравнению со стоимостью передачи объема данных, необходимых для установления транспортного соединения).



Дополнительная оптимизация достигается при использовании подпротокола управления сетевыми соединениями, который позволяет:

- мультиплексировать транспортные соединения в сетевые не только владельцу (то есть инициатору) сетевого соединения, как это делается в транспортном протоколе, но и партнеру по соединению;

- передать подробную информацию о причинах разъединения сетевых соединений с целью ускорить их восстановление;

- обеспечить идентификацию транспортных протоколов (как стандартных, так и нестандартных), используемых на данном сетевом соединении.

Эти задачи решаются процедурами подпротокола, в которых происходит обмен следующими протокольными блоками данных (ПБД): UN — ПБД использования соединения; NCM — ПБД управления соединением; DIAG — ПБД диагностики; NCMC — ПБД подтверждения управления соединением.

В ПБД UN указывается, как будет использоваться сетевое соединение — одним, либо последовательно несколькими транспортными протоколами и какими именно.

В ПБД NCM передается идентификатор сетевого соединения (это необходимо при его восстановлении), способ восстановления (то есть кто восстанавливает) и владелец. Владельцем может быть назначен объект — рецептор соединения. Кроме того, в NCM допускается указывать, что право на соединение имеют два объекта, тогда каждый из них может мультиплексировать свои транспортные соединения в данное сетевое соединение.

Блок DIAG служит для уточнения причины разъединения сетевого соединения (например, сетевое соединение больше не требуется, переполнение у рецептора и т.д.).

Блок NCMC содержит только код типа блока и служит ответом на ПБД NCM.

Порядок использования блоков сводится к следующему (рис. 3.28). При установлении сетевого соединения инициатор передает один из блоков UN, NCM или оба вместе. Если партнер согласен использовать протокол управления сетевыми соединениями, он отвечает блоком NCMC, если же отказывается, то иг-

норирует эти ПБД. Тогда сетевое соединение будет использоваться согласно стандартному транспортному протоколу.

Блок DIAG передается тогда, когда партнер не согласен установить сетевое соединение либо разъединяет его.

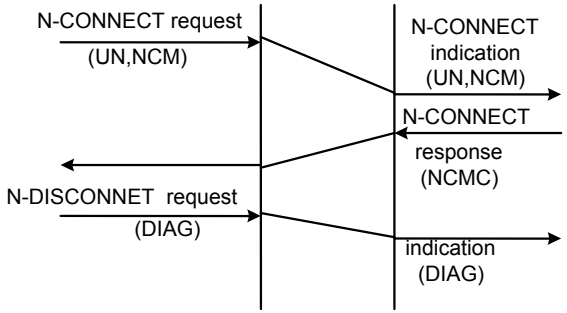



Рисунок 3.28 — Использование ПБД подпротокола управления сетевыми соединениями

Все ПБД передаются в полях данных пользователя сетевых сервисных примитивов, что отмечено скобками на рисунке. Рассматриваемый подпротокол имеет также средства обнаружения и устранения ситуаций дублирования сетевых соединений. Эти ситуации возникают вследствие равных прав на сетевое соединение, например, при его восстановлении. Устранение дублирования связано с ликвидацией одного из соединений.

При наличии сетевого сервиса без соединения транспортный сервис с соединением может быть обеспечен модифицированным транспортным протоколом класса 4.

Этот класс ориентирован на сетевой сервис типа С и имеет развитые средства защиты от ошибок, в том числе не обнаруживаемых сетевым уровнем. Как было показано, сетевой сервис без соединения отличается от сервиса с соединением перечнем и порядком использования сервисных примитивов, а также отсутствием извещения о потере данных. По своему качеству такой сервис может быть сравним с сервисом типа С. Для переориентации на другие сервисные примитивы существует модифицированный протокол класса 4. Эта модификация затрагивает



только те процедуры, которые связаны с доступом к сетевому соединению. Предоставляемый транспортный сервис при этом не меняется.

Рассмотрим теперь группу стандартов, ориентированных на предоставление транспортного сервиса без соединения. Такой сервис поддерживается протоколом типа «без соединения». Этот протокол может использовать оба типа сетевого сервиса за счет выбора одного из режимов передачи над сетевым сервисом: без соединения (РПБС) и с соединением (РПСС).

Протокол РПБС может быть охарактеризован как протокол минимальной функциональности: единственная функция, выполняемая протоколом — это защита транспортных ПБД от искажений с помощью контрольной суммы. Поскольку использование такой защиты определяется пользователем, в принципе возможен вариант, когда транспортный протокол ничего не добавляет к сетевому сервису, требуя тем не менее некоторых «накладных расходов» по обработке ТПБД. Существенным ограничением здесь являются максимальный размер блока передаваемых данных. Поскольку функция фрагментации отсутствует, сервисный блок (ТСБД) всегда упаковывается в один ТПБД (называемый UNITDATA), который передается в одном ССБД. Вследствие того, что длина последнего ограничена характеристиками сети передачи данных, ограничен и размер ТСБД.

Протокол с РПСС обладает большей функциональностью, чем протокол с РПБС. Здесь при поступлении каждого ТСБД производится установление сетевого соединения (или использования уже установленного соединения, если такое имеется), затем — передача единственного ТПБД, включающего весь ТСБД, и разъединение сетевого соединения (если необходимо).

Наличие сетевого соединения в протоколе РПСС не гарантирует отсутствия потери данных, даже если сетевой уровень работает без сбоев. Рассмотрим, в каких ситуациях такие потери возможны.

После того как сетевое соединение установлено, оба партнера получают равные права на передачу ТПБД и последующее разъединение сетевого соединения. Разъединение производится каждым партнером независимо и на основании собственного

алгоритма. Например, один из партнеров может разъединять соединение сразу после его использования, а другой — после ожидания следующих запросов на передачу ТСБД. Здесь возможна ситуация, когда ТПБД UNITDATA, отправляемый в поле данных сетевого сервисного примитива N-DATA request партнером Б, не достигает партнера А из-за того, что последний инициировал разъединение сетевого соединения после некоторого тайм-аута *T*.

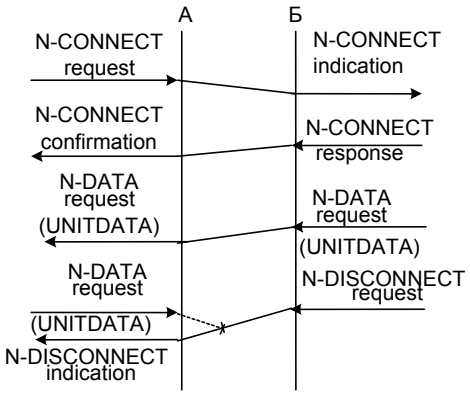



Рисунок 3.29 — Ситуация потери данных партнера А

Аналогичная ситуация возникает, когда разъединение инициируется рецептором (рис. 3.29). Здесь партнер А не может передать данные по устанавливаемому соединению потому, что партнер Б сразу после передачи своих данных разъединяет «чужое» соединение, не ожидая получения дополнительных запросов на передачу. Для защиты от таких ситуаций целесообразно предоставить право на разъединение только одному объекту — инициатору сетевого соединения.

Рассмотренные стандарты транспортного уровня включают широкий класс процедур, обеспечивающих единый транспортный сервис при использовании сетей передачи данных различного класса и с различным сервисом.



### 3.9.1 Транспортный сервис с соединением

Транспортный сервис этого типа обеспечивает передачу ТСБД по транспортному соединению. При этом пользователю предоставляются следующие возможности:

- адресация партнера. Установление транспортного соединения производится по адресу, указываемому пользователем. Этот адрес есть адрес транспортной сервисной точки доступа (ТСТД);

- выбор качества сервиса. Транспортный уровень необходим для оптимизации использования имеющихся сетевых ресурсов. Качество сервиса, требуемого взаимодействующими пользователями, обеспечивается при минимальной стоимости. Качество определяется выбором партнерами таких характеристик сервиса, как пропускная способность, транзитная задержка, коэффициент необнаруженных ошибок и вероятность отказов;

- использование разнообразных сетевых ресурсов. Предоставляемое пользователям транспортное соединение может быть основано на различных сетевых средствах, в том числе соединенных последовательно. Транспортный уровень скрывает от пользователей особенности сетевых средств;

- сквозная прозрачная передача ТСБД. Транспортный уровень обеспечивает соединение «из конца в конец». По этому соединению могут передаваться любые ТСБД с любым содержанием, форматом и способом кодирования. Содержание ТСБД не интерпретируется.

К транспортному сервису с соединением относятся следующие услуги: установление транспортного соединения; согласование КЧС; передача нормальных ТСБД; управление потоком ТСБД; передача срочных ТСБД; разъединение транспортного соединения. Эти услуги по своему назначению аналогичны услугам сетевого и канального уровней, рассмотренным ранее.

Соотношение сервисных примитивов на одном конце соединения. Все возможные последовательности сервисных примитивов могут быть описаны графом либо таблицей следования.

При простых соотношениях примитивов таблицы следования предпочтительнее, поскольку позволяют вводить понятие состояния КТС. Для рассматриваемого транспортного сервиса в такой таблице (таблица 3.4) знаком «+» отмечены допустимые последовательности.

Таблица 3.4

| Примитив снизу | Примитив справа | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | T-CONNECT request | T-CONNECT confirmation | T-CONNECT indication | T-CONNECT response | T-DATA request | T-DATA indication | T-EXPEDITED DATA request | T-EXPEDITED DATA indication | T-DISCONNECT request | T-DISCONNECT indication |
| T-CONNECT request | | + | | | | | | | + | |
| T-CONNECT confirmation | | | | | + | + | + | + | + | |
| T-CONNECT indication | | | | + | | | | | + | |
| T-CONNECT response | | | | | | | | | + | |
| T-DATA request | | | | | + | + | + | + | + | |
| T-DATA indication | | | | | + | + | + | + | + | |
| T-EXPEDITED DATA request | | | | | + | + | + | + | + | |
| T-EXPEDITED DATA indication | | | | | + | + | + | + | + | |
| T-DISCONNECT request | | | | | + | + | + | + | + | |
| T-DISCONNECT indication | | | | | + | + | + | + | + | |



Стандартом на транспортный сервис не регламентируется реакция на неправильную последовательность примитивов со стороны пользователя или транспортного уровня, поскольку это относится к локальным действиям, не затрагивающим взаимодействие объектов.

### 3.9.2 Качество транспортного сервиса

Под качеством транспортного сервиса понимаются характеристики услуг, составляющих сервис. Эти характеристики позволяют оценить вероятностно-временные параметры услуг с точки зрения пользователя сервиса, сформулировать требования к транспортному соединению и оценить, насколько эти требования удовлетворяются. Особенность транспортного сервиса заключается в том, что он «интегрирует» все нижележащие услуги. Поэтому параметры КЧС здесь связаны с параметрами сервиса нижележащих уровней и, в первую очередь, сетевого уровня.

Параметры КЧС разделяются на временные, вероятностные и прочие. Временные параметры характеризуют скорость ввода-вывода данных и время их передачи. Вероятностные параметры связаны с надежностью соединения. Прочие параметры связаны с использованием специальных механизмов транспортного уровня.

Вероятностно-временные параметры классифицируются по фазам соединения, то есть каждый из параметров связан с установлением соединения, передачей данных либо разъединением. Перечень используемых параметров КЧС приведен в таблице 3.5.

Таблица 3.5

| Фаза соединения | Параметры КЧС | | |
|---|---|---|---|
| | Временные | Вероятностные | Прочие |
| Установление | Задержка установления | Вероятность неустановления | |
| Передача данных | Пропускная способность, транзитная задержка | Коэффициент необнаруженных ошибок, живучесть, вероятность отказа | |
| Разъединение | Задержка разъединения | Вероятность неразъединения | |
| Независимо от фазы | | | Защита, приоритет |

По характеру использования эти параметры разделяются на согласуемые между пользователями, сообщаемые партнеру без согласования и не сообщаемые ни партнеру, ни транспортному уровню.

К согласуемым параметрам относятся пропускная способность и транзитная задержка, к несогласуемым, но сообщаемым партнеру, — коэффициент необнаруженных ошибок, защита и приоритет. Остальные параметры КЧС (вероятность неустановления, живучесть, вероятность отказа, вероятность неразъединения, задержка установления и задержка разъединения) не запрашиваются при обращении к транспортному уровню. Эти параметры необходимы пользователю для оценки адекватности своих протокольных механизмов. Параметры становятся известными пользователю либо априори (то есть до начала взаимодействия с транспортным уровнем), либо сообщаются в процессе использования транспортного сервиса при обмене служебными примитивами. Такие примитивы не рассматриваются как часть стандартного транспортного сервиса.

Как видно из таблицы 3.5, перечень параметров КЧС транспортного уровня совпадает с параметрами КЧС сетевого уровня, за исключением максимальной приемлемой стоимости. Описание каждого параметра не приводится, поскольку оно аналогично описанию, относящемуся к сетевому сервису, и легко получается из последнего простой заменой обозначений.

### 3.9.3 Транспортный сервис без соединения

Транспортный сервис без соединения используется для передачи отдельных независимых ТСБД. Каждый ТСБД передается или принимается при однократном обращении к сервису, описываемому следующими сервисными примитивами:

1) T_UNITDATA request — (вызывающий адрес, вызываемый адрес, параметры КЧС, данные пользователя);

2) T_UNITDATA indication: (вызывающий адрес, вызываемый адрес, параметры КЧС, данные пользователя).



Адресные параметры имеют те же значения, что и в сервисе с соединением. Под параметрами КЧС понимаются транзитная задержка, защита, коэффициент необнаруженных ошибок и приоритет.

Транспортный сервис без соединения в отличие от сервиса сетевого и канального уровней не содержит дополнительных возможностей взаимодействия пользователя и поставщика. Разработка этих вопросов предполагается в последующих редакциях стандарта.

## 3.10 Сервис сеансового уровня

Сеансовый сервис обеспечивает прикладные объекты следующими средствами равноправного, синхронизированного, структурированного взаимодействия:

- установления сеансового соединения, синхронизированного обмена данными, упорядоченного и безусловного завершения сеансового соединения;
- согласования использования маркеров обмена данными, синхронизации и завершения взаимодействия, а также фиксации маркеров на одной из взаимодействующих сторон;
- установления точек синхронизации внутри диалога;
- выполнения ресинхронизации сеансового соединения к согласованной прикладными объектами точке синхронизации;
- прерывания диалога и его возобновления с заранее организованной точки синхронизации.

Сеансовый сервис обеспечивается сеансовым протоколом, который, в свою очередь, использует сервис, предоставляемый транспортным уровнем. На сеансовом уровне вводится ряд основополагающих понятий, которые широко используются при описании объектов представительного и прикладного уровней.

К таким понятиям относятся маркер, точка синхронизации, диалоговый элемент, активность, ресинхронизация, переговоры, функциональная группа, качество сеансового сервиса и др.

*Маркер*

Маркер — это атрибут сеансового соединения, который динамически назначается одному из взаимодействующих при-

кладных объектов (SS-пользователю). Владелец маркера имеет исключительное право инициировать выполнение услуги, контролируемой данным маркером.

На сеансовом соединении могут быть использованы четыре маркера:

1. DK — маркер данных (Data Token).
2. TR — маркер завершения (Release Token).
3. MI — маркер малой синхронизации (Synhronize Minor Token).
4. MA — маркер большой синхронизации (Major-activityToken).

Каждый маркер на сеансовом соединении всегда находится в одном из двух состояний: доступен и недоступен. Маркер доступен тогда, когда SS-пользователи, в ходе установления сеансового соединения, согласовали его применение в процессе предстоящего взаимодействия. В этом случае маркер назначается одной из взаимодействующих сторон, а поставщик сеансового сервиса гарантирует отсутствие раздвоения маркера в любой момент существования сеансового соединения. Маркер недоступен тогда, когда в ходе установления сеансового соединения SS-пользователи не включили его в число доступных маркеров, применяемых в процессе предстоящего взаимодействия. При этом маркер не может быть назначен ни одной из взаимодействующих сторон.

*Точка синхронизации*

При передаче данных пользователи сеансового сервиса могут размещать в потоке данных точки синхронизации. Точки синхронизации идентифицируются последовательными монотонно возрастающими номерами, которые обеспечиваются поставщиком сеансового сервиса. Взаимодействующие прикладные объекты могут связывать с точками синхронизации, а точнее, с их номерами, любую семантику, определяемую приложением.

Фиксацию точек синхронизации можно выполнять двумя способами, используя технику малой или большой синхронизации. В зависимости от того, каким способом формируется точка синхронизации, она будет называться либо точкой большой синхронизации (major synchronization), либо точкой малой синхронизации (minor synchronization). Поставщик сеансового сервиса нумерует точки синхронизации последовательно независимо от того, являются ли они точками большой или малой синхронизации (рисунок 3.30,*а*). Отличие заключается в механизме выполнения фиксации точек синхронизации. Точка большой синхронизации считается зафиксированной после ее явного подтверждения, в то время как точка малой синхронизации для своей фиксации не требует явного подтверждения и может вообще не подтверждаться.



*Диалоговый элемент*

С понятием точки синхронизации связано понятие диалогового элемента. Точки большой синхронизации используются для структурирования обмена данными в виде серии диалоговых элементов. Точка большой синхронизации завершает один диалоговый элемент и начинает другой. Диалоговый элемент — логически выделенное замкнутое взаимодействие, которое можно отработать независимо от предшествующих и последующих процессов обмена данными. Точки малой синхронизации используются для структуризации обмена данными в рамках диалогового элемента, являясь отметками прохождения последовательно отрабатываемых этапов распределенного взаимодействия партнеров.

*Активность*

Последовательность диалоговых элементов может выступать в качестве более крупной логически завершенной части работы, называемой активностью (рис. 3.30,*б*). В рамках сеансового соединения одновременно может существовать лишь одна активность. Допускается последовательная обработка активностей на одном сеансовом соединении. В то же время активность может охватывать несколько последовательно устанавливаемых сеансовых соединений. Это связано со свойствами прерываемости активности, независимой идентификации активности в рамках поставщика сеансового сервиса, а также со свойством возобновляемости прерванной активности на любом подходящем сеансовом соединении.


*а*


*б*

Рисунок 3.30
*а* — диалоговый элемент и *б* — активность

*Ресинхронизация*

Является средством принудительной координации взаимодействующих сторон в рамках сеансового соединения. Ресинхронизация может быть инициирована любым SS-пользователем в любой момент существования сеансового соединения. При этом соединение переходит в определенное состояние, позволяющее перераспределить маркеры между сторонами и согласовать новое значение последовательного номера точки синхронизации для продолжения взаимодействия. В ходе ресинхронизации осуществляется сброс всех данных, находящихся в это время в стадии передачи по сеансовому соединению. Поставщик сеансового сервиса обеспечивает три режима согласования нового значения последовательного номера точки синхронизации, выбирается ранее не задействованное значение — отказ (abandon); любое значение, которое больше, чем номер последней подтвержденной точки большой синхронизации, и не больше, чем последний полученный номер точки синхронизации — рестарт (restart); любое значение из разрешенного диапазона, указанное прикладным объектом, — установка (set).



*Функциональные группы и стандартные подмножества сеансового сервиса*

Функциональная группа — это объединение логически связанных услуг. Функциональные группы вводятся с целью согласования требований пользователей в ходе фазы установления сеансового соединения. В том случае, когда какая-либо услуга, входящая в функциональную группу, контролируется маркером, в функциональную группу включаются услуги управления расстановкой маркеров.

Базовая функциональная группа объединяет основные сеансовые услуги, позволяющие установить сеансовое соединение, осуществить передачу нормальных блоков данных и завершить сеансовое соединение, то есть базовую функциональную группу образуют следующие услуги: S-CONNECT, S-DATA, S-RELEASE, S-U-ABORT и S-P-ABORT.

Функциональная группа согласованного завершения содержит услугу S-RELEASE, контролируемую маркером завершения. Эта функциональная группа обеспечивает дополнительное свойство услуге упорядоченного завершения сеансового соединения, при котором сеанс прекращается только при обоюдном согласии партнеров. Функциональная группа включает услуги S-RELEASE, S-GIVE-TOKENS и S-PLEASE-TOKENS.

Функциональная группа полудуплекса определяет режим передачи данных. Владелец маркера данных имеет право фор-

мировать нормальный поток NSSDU. Функциональная группа объединяет услуги S-GIVE-TOKENS и S-PLEASE-TOKENS.

Функциональная группа дуплекса поддерживает дуплексный нормальный поток блоков данных NSSDU. Оба пользователя имеют равные права инициировать услугу S-DATA. Функциональная группа не содержит никаких услуг. Недопустимо одновременное использование в рамках сеансового соединения функциональных групп дуплекса и полудуплекса.

Функциональная группа срочных данных содержит услугу S-EXPEDITED-DATA, позволяющую формировать срочный поток блоков данных XSSDU.

Функциональная группа типизированных данных предоставляет услугу образования пользователями потока типизированных блоков данных TSSDU. Содержит услугу S-TYPED-DATA.

Функциональная группа обмена данными обеспечивает услугу S-CAPABILITY-DATA. Группа может быть выбрана пользователем лишь вместе с группой управления активностью.

Функциональная группа малой синхронизации включает в свой состав следующие услуги: S-SYNC-MINOR, S-GIVE-TOKENS и P-PLEASE-TOKENS.

Функциональная группа большой синхронизации включает в свой состав услуги S-SYNG-MAJOR, S-GIVE-TOKENS и P-PLEASE-TOKENS.

Функциональная группа ресинхронизации представляет услугу S-RESYNCHRO-NIZE.

Функциональная группа оповещения обеспечивает услуги S-U-EXCEPTION-REPORT и S-P-EXCEPTION-REPORT.

Функциональная группа управления активностью включает все услуги управления активностью и услугу передачи управления: S-ACTIVITY-START, S-ACTIVITY-RESUME, S-ACTIVITY-INTERRUPT, S-ACTIVITY-DISCART, S-ACTIVITY-END, S-GIVE-TOKENS, S-PLEASE-TOKENS и S-GIVE-CONTROL.

Свойство маркерного управления определенной услуги передается функциональной группе, содержащей эту услугу. Ниже



приводятся функциональные группы, контролируемые маркерами.

| | |
|---|---|
| Согласованное завершение | TR |
| Полудуплекс | DK |
| Малая синхронизация | Ml |
| Большая синхронизация | MA |
| Управление активностью | MA |

При использовании функциональных групп, требующих управления маркерами, прикладной протокол в ходе установления сеансового соединения обязан объявить соответствующий маркер доступным на сеансовом соединении и указать его начальную расстановку.

В стандарте вводится понятие сервисного подмножества (сервисного профиля сеансового соединения), обеспечиваемого на сеансовом соединении. Сервисное подмножество — это комбинация базовой функциональной группы с другими функциональными группами при условии, что функциональная группа обмена данными включается в подмножество лишь наряду с функциональной группой управления активностью, а функциональная группа оповещения включается в подмножество лишь наряду с функциональной группой полудуплекса.

Каждая сеансовая услуга представляется набором примитивов. Отработка определенной сеансовой услуги заключается в последовательном исполнении ее примитивов.

Различают услуги двух типов: подтверждаемые и неподтверждаемые. Подтверждаемая услуга содержит четыре примитива: запрос (request), индикация (indication), ответ (response), подтверждение (confirmation). Неподтверждаемая услуга содержит два примитива: запрос и индикация, а в специальных случаях — только индикация.

Услуга установления сеансового соединения, применяемая в соответствующей фазе, состоит из четырех примитивов: S-CONNECT request, S-CONNECT indication, S-CONNECT response, S-CONNECT confirmation. Тип услуги подтверждаемый. В таблице 3.6 приведен состав сеансовых услуг фаз передачи данных и завершения соответственно.

Таблица 3.6

| Услуга | Примитивы | Тип услуги |
|---|---|---|
| Передача нормальных данных | S-DATA (request, indication) | Неподтверждаемая |
| Передача срочных данных | S-EXPEDITED-DATA (request, indication) | Неподтверждаемая |
| Передача типизированных данных | S-TYPED-DATA (request, indication) | Неподтверждаемая |
| Услуга обмена данными | S-CAPABILITY-DATA (request, indication, response, confirmation) | Подтверждаемая |
| Передача маркеров | S-GIVE-TOKENS (request, indication) | Неподтверждаемая |
| Услуга требования маркеров | S-PLEASE-TOKENS (request, indication) | Неподтверждаемая |
| Передача управления | S-GIVE-CONTROL (request, indication) | Неподтверждаемая |
| Малая синхронизация | S-SINC-MINOR (request, indication, response, confirmation) | Подтверждаемая |
| Большая синхронизация | S-SINC-MAJOR (request, indication, response, confirmation) | Подтверждаемая |
| Ресинхронизация | S-RESYNCHRONIZE (request, indication, response, confirmation) | Подтверждаемая |
| Услуга оповещения поставщика | S-P-EXCEPTION-REPORT (indication) | Неподтверждаемая |
| Услуга оповещения пользователя | S-U- EXCEPTION-REPORT (request, indication) | Неподтверждаемая |
| Инициация активности | S-ACTIVITY-START (request, indication) | Неподтверждаемая |
| Услуга возобновления активности | S-ACTIVITY-RESUME (request, indication) | Неподтверждаемая |
| Услуга прерывания активности | S-ACTIVITY-INTERRUPT(request, indication, response, confirmation) | Подтверждаемая |
| Сброс активности | S-ACTIVITY-DISCARD (request, indication, response, confirmation) | Подтверждаемая |



Окончание табл. 3.6

| Услуга | Примитивы | Тип услуги |
|---|---|---|
| Завершение активности | S-ACTIVITY-END (request, indication, response, confirmation) | Подтверждаемая |
| Упорядоченное завершение | S-RELEASE (request, indication, response, confirmation) | Подтверждаемая |
| Безусловное завершение пользователя | S-U-ABORT (request, indication) | Неподтверждаемая |
| Безусловное завершение поставщика | S-P-ABORT (indication) | Неподтверждаемая |

*Услуга малой синхронизации.* Услуга малой синхронизации S-SYNC-MINOR обеспечивает SS-пользователей средствами фиксации точек малой синхронизации в потоках нормальных и типизированных данных на сеансовом соединении. В случае, когда на сеансовом соединении выбрана для использования функциональная группа управления активностью, инициация услуги малой синхронизации должна осуществляться лишь в рамках существующей активности. Услуга малой синхронизации является контролируемой маркером, право отработать примитив S-SYNC-MINOR request имеет тот участник взаимодействия, который владеет маркером малой синхронизации и маркером данных, если последний доступен на сеансовом соединении. Услуга малой синхронизации является подтверждаемой. Однако поставщик сеансового сервиса специально не ожидает инициации примитива S-SYNC-MINOR response вслед за отработанным им примитивом S-SYNC-MINOR indication даже в том случае, когда подтверждение точки малой синхронизации затребовано явно. Следующая инициация услуги S-SYNC-MINOR может быть выполнена SS-пользователем до того, как будет обработан сервисный примитив S-SYNC-MINOR confirmation, на предыдущий запрос малой синхронизации. Таким образом, в определенный момент на сеансовом соединении может существовать целый ряд неподтвержденных точек малой синхронизации. При этом число неподтвержденных точек синхронизации, фиксируемых с помощью услуги малой синхронизации, не ограничивается поставщиком сеансового сервиса. Сервисный примитив S-SYNC-MINOR confirmation осуществляет подтверждение всех малых точек син-

хронизации вплоть до той, последовательный номер которой указан в самом примитиве.

Рассмотрим один возможный сценарий фиксации синхроточки с помощью услуги малой синхронизации (рис. 3.31).

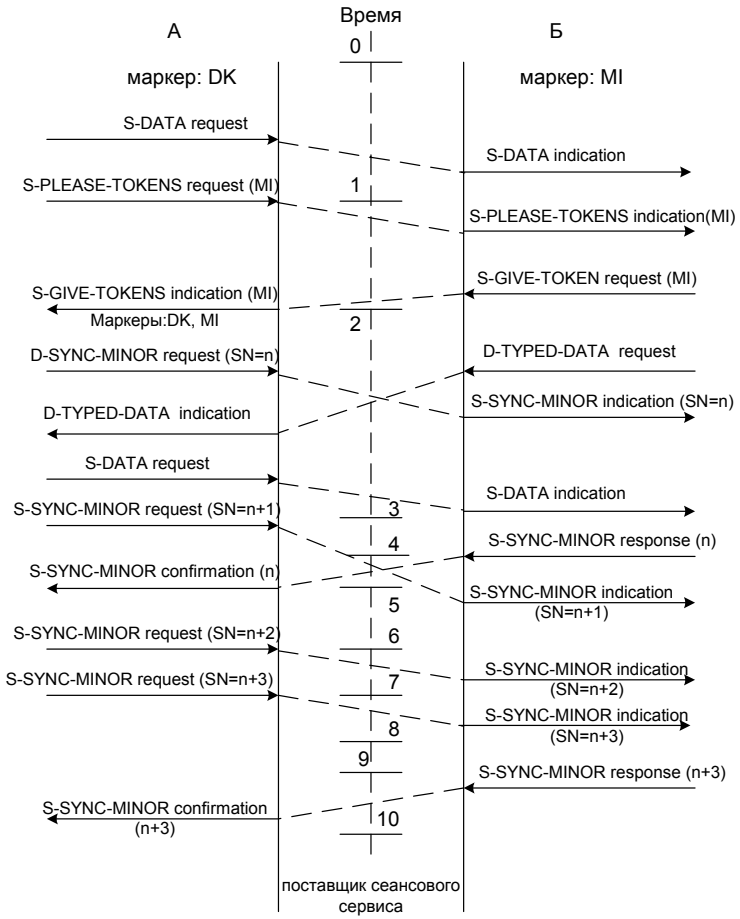



Рисунок 3.31 — Фиксация синхроточек с помощью услуги малой синхронизации



Пусть к моменту рассмотрения существует следующая расстановка маркеров: на стороне А расположен маркер DK, на стороне Б — маркер Ml. В соответствии с прикладным протоколом в момент 1 SS-пользователю на стороне А необходимо выполнить фиксацию синхроточки с помощью услуги малой синхронизации. Чтобы удовлетворить маркерным ограничениям, на стороне А инициируется примитив S-PLEASE-TOKENS request с требованием маркера Ml. Отработка индикации услуги требования маркеров на стороне Б влечет передачу маркера Ml на сторону А посредством инициации примитива S-GIVE-TOKENS request. Таким образом, осуществляется перераспределение маркеров между сторонами А и Б. В результате, начиная с момента 2, SS-пользователь на стороне А имеет право инициировать услугу малой синхронизации, что и происходит, причем параметр «тип» примитива S-SYNC-MINOR request указывает на необходимость подтвердить фиксируемую синхроточку явным образом (на рисунке не показано).

В ходе отработки примитивов запроса и индикации на обеих сторонах фиксируемая синхроточка получает последовательный номер SN, равный *n*. До получения подтверждения синхроточки с номером *n* в момент 3 со стороны А начинается фиксация синхроточки, последовательный номер которой будет *n*+1. На стороне Б SS-пользователь в момент 4 инициирует сервисный примитив S-SYNC-MINOR response, подтверждающий синхроточку с номером *n,* о чем становится известно на стороне А в момент 5. При этом на сеансовом соединении остается одна неподтвержденная синхроточка с номером SN = *n*+1. В моменты 6 и 7 фиксируются синхроточки с последовательными номерами SN=n+2 и SN = n+3 соответственно, так что к моменту 8 на сеансовом соединении будут существовать три неподтвержденные синхроточки с номерами *n*+1, *n*+2, *n*+3. На стороне Б SS-пользователь инициирует сервисный примитив S-SYNC-MINOR response, кумулятивно подтверждающий все синхроточки с номерами до n + 3 включительно. Таким образом, с момента 10 на сеансовом соединении не существует неподтвержденных точек малой синхронизации.

*Услуга большой синхронизации*

Услуга большой синхронизации S-SYNC-MAJOR обеспечивает SS-пользователей средствами фиксации точек большой синхронизации в потоках нормальных, типизированных и срочных данных с целью полного отделения потоков данных, существовавших до момента фиксации синхроточки, от потоков данных, организуемых после момента фиксации синхроточки.

Рассмотрим сценарий фиксации синхроточки с помощью услуги большой синхронизации (рис. 3.32).

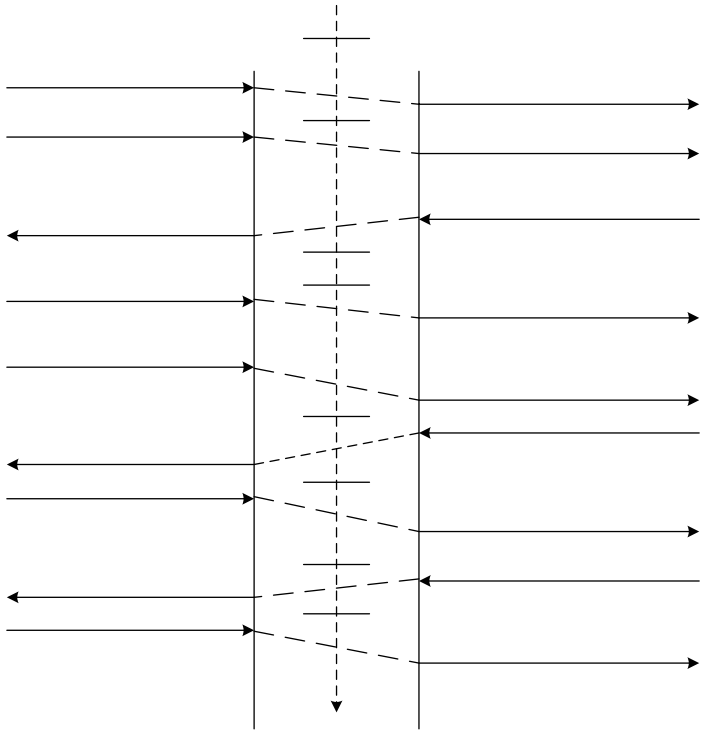


Рисунок 3.32 — Фиксация синхроточки с помощью услуги большой синхронизации



Пусть к моменту рассмотрения существует следующая расстановка маркеров: на стороне А расположен маркер DK, на стороне Б — маркеры М1 и МА. Для фиксации точки малой синхронизации SS-пользователь на стороне А в момент 1 инициирует услугу требования маркера малой синхронизации М1. Получив маркер М1 в момент 2, SS-пользователь на стороне А отрабатывает в момент 3 сервисный примитив S-SYNC-MINOR request.

При этом на обеих сторонах фиксируется точка малой синхронизации с номером SN, равным *n*. Для того чтобы теперь выполнить фиксацию синхроточки с помощью услуги большой синхронизации, на стороне А должен находиться наряду с имеющимися DK и MI маркер МА. К моменту 4 маркер МА располагается на стороне А, что позволяет в момент 5 инициировать сервисный примитив S-SYNC-MAJOR request, в результате отработки которого на обеих сторонах сеансового соединения фиксируется точка большой синхронизации с номером SN, равным *n*+1. В момент 6 SS-пользователь отрабатывает сервисный примитив S-SYNC-MAJOR response, осуществляя, таким образом, кумулятивное подтверждение точек малой синхронизации с номерами до *n* включительно и точки большой синхронизации с номером *n*+1. С момента 7 на сеансовом соединении не существует неподтвержденных точек синхронизации.

*Услуга ресинхронизации.* Услуга ресинхронизации S-RESYNCHRONIZE применяется SS-пользователями для упорядоченного восстановления синхронизированного взаимодействия в рамках сеансового соединения, которое было нарушено из-за ошибок SS-пользователей или поставщика сеансового сервиса, не приведших к безусловному завершению соединения. Услуга ресинхронизации может быть инициирована любым из участников синхронизированного взаимодействия в любой момент существования сеансового соединения. После инициации сервисного примитива S-RESYNCHRONIZE request инициатор не имеет права начать никакую другую услугу, кроме S-U-ABORT, до момента получения сервисного примитива S-RESYNCHRONIZE confirmation.

При ресинхронизации осуществляется чистка сеансового соединения, сбрасываются все данные, находящиеся в процессе передачи. Услуга ресинхронизации позволяет SS-пользователям

осуществить расстановку маркеров, доступных на сеансовом соединении, а также осуществить выбор точки синхронизации для продолжения взаимодействия.

Если до момента начала ресинхронизации на сеансовом соединении существовали неподтвержденные точки синхронизации, они остаются неподтвержденными. При этом после завершения ресинхронизации прерванные услуги фиксации синхроточек не продолжаются, но могут быть инициированы в установленном порядке.

## 3.11 Сервис представительного уровня

Оконечные системы (абоненты) вычислительных сетей весьма разнообразны и представлены устройствами различных типов от простых символьно-ориентированных дисплеев до универсальных ЭВМ и систем, ориентированных на базы данных. В разных устройствах используется различное внутреннее представление хранимой информации. Для их взаимодействия модель ВОС содержит шестой уровень, называемый уровнем представления, или представительным уровнем.

В процессе своего функционирования объекты уровня представления используют протокол, который позволяет согласовать различия в синтаксисе данных с помощью преобразования локального синтаксиса (внутреннее представление) в стандартный синтаксис передачи. Синтаксис передачи вместе с правилами преобразования составляет образ конкретного соединения уровня представления. Согласование синтаксических различий должно выполняться таким образом, чтобы сохранить семантику передаваемой информации.

В настоящее время разработаны стандарты на сервис и протокол представительного уровня, которые позволяют создать достаточно общие и мощные средства, не зависящие от конкретных реализаций и интерфейсов, используемых в различных вычислительных системах.

Представительный уровень в процессе согласования синтаксических различий имеет дело с двумя аспектами. Первый аспект относится к заданию (описанию) синтаксиса со стороны прикладных объектов, а второй — к тому, как описанные заданным



синтаксисом данные выражаются в терминах представления их значений в окружении ВОС. Первый аспект обозначается термином абстрактный синтаксис, а второй — синтаксис передачи.

Абстрактный синтаксис предоставляет возможность прикладным системам выполнять спецификацию передаваемых данных способом, не зависящим от конкретных методов кодирования, используемых для представления данных. Абстрактный синтаксис может быть описан множеством способов. Наиболее широко используется Международный стандарт, определяющий понятия абстрактного синтаксиса ASN.1, который используется, в частности, для задания синтаксиса при определении сервиса и протоколов верхних уровней, разрабатываемых в МОС. Таким образом, абстрактный синтаксис является атрибутом прикладного уровня и не рассматривается в представительном протоколе. Представительный уровень имеет дело только с идентификацией, которая должна обеспечивать уникальный доступ к тому или иному абстрактному синтаксису.

Синтаксис передачи определяет правила кодирования, которые задают специфику представления данных во время их передачи между открытыми системами. Он имеет дело со способом, которым фактически представляются эти данные в виде последовательностей нулей и единиц (бит). Здесь также предполагается существование множества синтаксисов передачи. Одним из примеров может служить стандарт МОС. Сам представительный протокол никак не определяет такие правила кодирования, а имеет дело только с использованием.

Хотя более правильным является термин «уровень представления», часто используется название «представительный уровень» по аналогии со всеми другими уровнями — транспортным, сеансовым, физическим и др.

Конкретная семантика прикладных систем может быть выражена с помощью различных абстрактных синтаксисов ($АС_1$, ..., $АС_n$), каждый из которых использует одинаковые или различные синтаксисы передачи ($СП_1$, ..., $СП_m$) (рис. 3.33,*а*). Из рисунка видно, что возможны различные способы выражения семантики абстрактными синтаксисами и абстрактных синтаксисов синтаксисами передачи. Так, различная семантика может использовать различный (рис. 3.33,*б*) или одинаковый (рис. 3.33,*в*) абстракт-

ный синтаксис с одинаковым синтаксисом передачи. Для указания используемой комбинации абстрактного синтаксиса и синтаксиса передачи используется термин «контекст представления». Работа с контекстом представления поддерживается представительным сервисом.

В общем, не предполагается однозначного соответствия между конкретными абстрактными синтаксисами и синтаксисами передачи. Сервис представительного уровня позволяет устанавливать такие соотношения динамически. Например, передача с использованием одного и того же абстрактного синтаксиса может происходить с шифрацией или без шифрации, со сжатием или без сжатия передаваемой информации. Кроме того, стандарт прикладного уровня позволяет использовать один и тот же синтаксис в качестве как абстрактного, так и синтаксиса передачи, но в этом случае нет возможности задать альтернативные синтаксисы передачи.

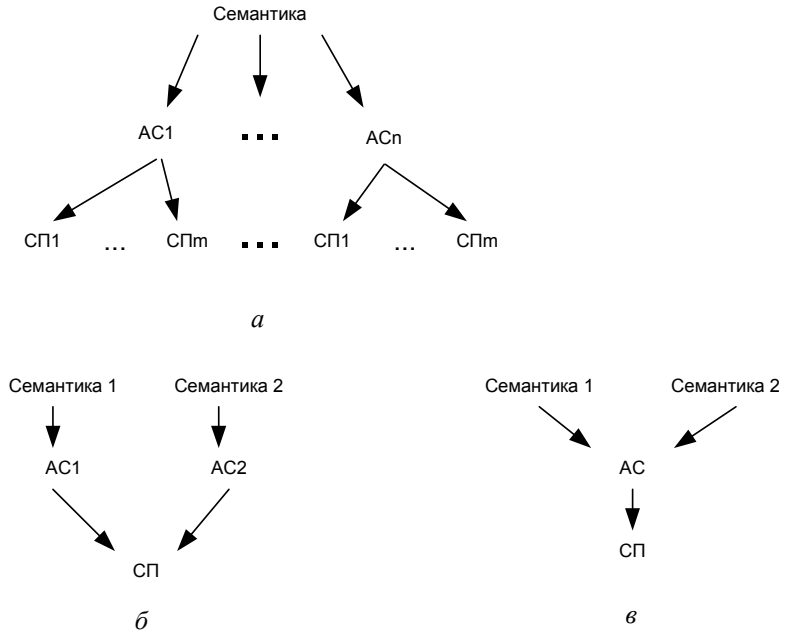



Рисунок 3.33 — Связь семантики, абстрактных синтаксисов и синтаксисов передачи системы идентификаций синтаксисов передачи



Рассматриваемый стандарт представительного уровня обеспечивает средства для задания представительных контекстов на этапе установления представительного соединения и на существующем соединении. Для задания представительного контекста пользователь представительного сервиса идентифицирует абстрактный синтаксис, в то время как поставщик сервиса согласует синтаксис передачи, который поддерживает этот абстрактный синтаксис в процессе взаимодействия между открытыми системами. Для такого согласования поставщик сервиса использует специальный механизм, называемый переговорами.

Как минимум один контекст представления должен существовать на представительном уровне. Он необходим хотя бы для того, чтобы данные о переговорах можно было трактовать в каком-либо контексте. Такой контекст называется контекстом умолчания и может быть задан либо явным его указанием при установлении соединения, либо неявным способом вне заданного стандарта.

На соединении в одно и то же время может быть задано несколько контекстов. Заданные в данный момент контексты на представительном соединении образуют множество заданных контекстов, которые могут в любом сочетании использовать один и тот же или разные абстрактные синтаксисы и/или один и тот же или разные синтаксисы передачи. В этом случае типы данных пользователей идентифицируются одним из представительных контекстов, применяемых к значениям данных, и синтаксическим описанием значений данных в этом контексте. Значения данных могут быть структурированы так, чтобы содержать значения вложенных данных из других заданных контекстов представления.

Ответственность за выбор и согласование множества используемых абстрактных синтаксисов несут взаимодействующие прикладные объекты. Они должны информировать представительный уровень об этих соглашениях посредством указания имен абстрактных синтаксисов. Зная множество абстрактных синтаксисов, используемых прикладными объектами, объекты представительного уровня отвечают за выбор совместно используемых синтаксисов передачи. Таким образом, можно выделить три важные функции представительного уровня:

- согласование синтаксисов передачи;

- преобразование в соответствии с синтаксисом передачи;
- преобразование из синтаксиса передачи.

Функция согласования синтаксисов передачи поддерживается механизмом переговоров, выполняемым в соответствии с представительным протоколом, а преобразования в синтаксис передачи и обратно выполняются внутри представительного объекта невидимым со стороны представительного протокола способом.

### 3.11.1 Спецификация синтаксиса для верхних уровней

На нижних уровнях эталонной модели ВОС природа параметров заголовков протокольных и интерфейсных элементов достаточна проста и позволяет рассматривать их как последовательность бит. С переходом на более высокие уровни ситуация изменяется. Так, уже на сеансовом уровне заголовок имеет сложную структуру, содержащую набор различных параметров, часть которых может объединяться в различные группы. Параметры и группы бывают обязательными и необязательными, а необязательные, в свою очередь, определяются или не определяются умолчанием. Сложная природа параметров и их групп требует введения механизмов контроля, как наличия самих параметров, так и их содержимого.

Ситуация еще более усложняется на представительном уровне. Данные, передаваемые на этом уровне, могут содержать структуры сложных типов, определяемых системами прикладного уровня, и, возможно, включающие строки из различных символьных множеств. Для задания таких структур необходимо разработать набор правил, используя которые, можно определять синтаксис структур и составляющих их типов. Набор таких правил, созданных в рамках ВОС, носит название «понятие абстрактного синтаксиса» (ASN.l -Abstract Syntax Notation One).

Система абстрактного синтаксиса дополняется правилами кодирования, определяющими значения потока октетов. В принципе, спецификации на ASN.1 могут быть представлены различными способами кодирования.

Целью спецификаций средствами ASN.1 является строгое определение правил формирования информационного потока из произвольных достаточно сложных значений и выделения этих



значений при обработке потока. Центральную роль в данном процессе играет понятие типа.

Множества однородных значений характеризуются типом, а конкретного представителя из таких множеств будем называть значением соответствующего типа.

Разработчики ASN.1, прежде всего, выделили базовый набор простых типов, которые иногда называют встроенными. К ним относятся булевские (BOOLEAN), целые (INTEGER), битовые строки (BITSTRING), строки октетов (OCTETSTRING) и специальный пустой тип (NULL).

Тип BOOLEAN представляет множество, состоящее из двух элементов, характеризующих значения этого типа, — TRUE (истина) и FALSE (ложь).

Тип INTEGER включает все положительные и отрицательные значения целых чисел.

Тип BITSTRING содержит битовые строки, представляющие произвольные последовательности нулей и единиц. Описываемый стандарт не накладывает никаких ограничений на размер этих строк.

Тип OCTETSTRING рассматривает строки как произвольную последовательность восьмибитовых значений (от 0 до 255). В этом случае ограничений на длину строк также не существует.

Тип NULL состоит из одного элемента NULL, который характеризует пустое значение. В таблице 3.7 приведены простые типы и определяемые ими значения.

Таблица 3.7

| Простой тип | Значения |
|---|---|
| BOOLEAN | TRUE, FALSE |
| INTEGER | Множество положительных и отрицательных чисел |
| BITSTRING | Произвольная последовательность нулей и единиц |
| OCTETSTRING | Последовательность октетов, содержащих произвольные значения в диапазоне от 0 до 255 |
| NULL | NULL |

Используя встроенные типы и применяя небольшой набор способов построения, можно получать сложные типы. Способы можно применять в любой последовательности, повторно, многократно и не только к встроенным, но и к полученным предыдущим применением способов сложным типам. Полученные в результате сложные типы называют структурированными. Существуют пять способов структурирования:

- списком,
- списком из,
- множеством,
- множеством из,
- выбором и вырезкой.

При формировании списком (SEQUENCE) задан список (упорядоченный набор) имеющихся типов. Конкретное значение для такого списка может быть получено упорядоченной последовательностью значений, одного для каждого типа из списка. Набор всех возможных значений, получаемых таким способом, образует новый тип.

При формировании множеством (SET) задан неупорядоченный набор различных существующих типов. Конкретное значение для такого набора может быть получено как неупорядоченное множество значений — одного для каждого представленного типа. Набор всех возможных значений, полученный таким способом, образует новый тип.

При формировании списком из (SEQUENCE OF) задан один из существующих типов. Значение можно формировать в виде упорядоченной последовательности нулевого, одного или большего числа значений данного типа. Набор (возможно, бесконечный) всех значений, получаемых таким способом, становится новым типом.

При формировании множеством из (SET OF) задан один из существующих типов. Значение можно формировать в виде неупорядоченного множества значений данного типа. Набор всех значений (возможно, бесконечный), получаемых таким способом, становится новым типом.

При формировании выбором (CHOICE) задан список различных существующих типов. Значение можно получить выбором одного из них. Множество всех возможных значений, получаемых таким способом, образует новый тип.



При формировании вырезкой (SELECTION) задан один из существующих типов. Новый тип можно получать как подмножество, используя некоторую структуру или отношение порядка возможных значений.

И, наконец, могут встретиться типы, множество значений которых определено иначе. Такие типы называются внешними.

Каждый тип, используемый в стандарте ASN.1, содержит специальный признак, называемый тегом. Тег определяется либо самим стандартом, либо пользователем. Тег вводится скорее для машинного использования, чем для человека. Один и тот же тег может назначаться различным типам. В этом случае теги будут трактоваться в соответствии с текущим контекстом. В то же время для обозначения одного и того же типа пользователь может использовать различные теги. В последнем случае типы являются изоморфными, но отличаются друг от друга.

Тег состоит из идентификатора класса и неотрицательного целого, различающего теги внутри класса. Определено четыре класса тегов:

UNIVERSAL — используется только для данных, представленных в соответствии с правилами ASN.1 и назначается либо одному из типов, либо одному из способов формирования сложных типов. В таблице 3.8 перечислены теги этого класса.

Таблица 3.8

| Теги класса | Значение | Теги класса | Значение |
|---|---|---|---|
| 0 | Зарезервирован | 6. ..15 | Не используется |
| 1 | BOOLEAN | 16 | Формирование списком или выбором |
| 2 | INTEGER | 17 | Формирование множеством или вырезкой |
| 3 | BITSTRING | 18. . .22,25 | Типы строк символьного множества |
| 4 | OCTETSTRING | 23,24 | Типы времени |
| 5 | NULL | 26 и т. д. | Не используются |

APPLICATION — применяется для определения типов, построенных по правилам других стандартов. Каждый тег из этого класса служит для обозначения только одного типа;

PRIVATE — используется для обозначения типов, не входящих ни в один из стандартов;

класс контекстно-зависимых тегов служит для интерпретации типов в зависимости от контекста.

*Описания типов*

Абстрактный синтаксис ASN.1 представляет собой средство со строгой типизацией, что требует явного описания используемых типов. Одним из простейших типов в ASN.1, позволяющих представлять логические значения, является булевский. Для объявления имени этого типа используется служебное слово BOOLEAN, например, объявление:

холостой : := BOOLEAN

означает, что переменные с типом «холостой» могут содержать одно из двух значений — TRUE или FALSE.

Подобным же образом определяется тип для целых значений, например:

табельный номер : := INTEGER.

Дополнительно средствами ASN.1 можно задавать диапазоны или множества допустимых целых чисел:

деньмесяца : := INTEGER {first (1), last(31)}

деньнедели : := INTEGER {понедельник (0), вторник (1), среда (2), четверг (3), I пятница (4), суббота (5), воскресенье (6)}

Первый из примеров представляет тип переменной, которая может принимать любое целое значение из диапазона с минимальным значением 1 и максимальным значением 31, а второй позволяет моделировать значения переменных с несколькими состояниями.

Тип OCTETSTRING используется для моделирования двоичных данных. Используя этот тип, можно описывать строки неспецифицированного формата и длины:

Факсимильный : := BITSTRING

-- последовательность бит в соответствии с рекомендацией Т.4МККТТ

Здесь и далее строка, начинающаяся с двух символов «-», означает комментарий.



Можно указывать строки фиксированных размеров:

Фиксированная строка : : = BITSTRING {first (1), last (20)}.

И, наконец, можно описывать строки, в которых содержатся логические переменные:

Радуга : := BITSTRING {

красный (0), оранжевый (1), желтый (2), зеленый (3), голубой (4)}

Тип OCTETSTRING также определяет двоичные данные, число бит в которых кратно 8:

Факсимильный : := OCTETSTRING

--последовательность октетов в соответствии с рекомендацией Т.5 и Т.6 МККТТ

Значения, которые содержат переменные с типами BITSTRING и OCTETSTRING, могут представляться в 16-ричном

'A98A' H

или двоичном

'1010100110001010' B

виде. Оба приведенных здесь представления эквивалентны.

Особо выделяются типы символьных и числовых строк. Для символьных строк, соответствующих стандарту, используется тип ISO646Strmg, а для числовых NumeringString. Оба типа относятся к классу UNIVERSAL, причем первый имеет номер 22, а второй 18.

Аналогично введены типы строк печатаемых символов [UNIVERSAL, 19] с именем PrintableString, строк телексных символов [UNIVERSAL, 20] с именами Telex-String и T61 String, строк символов видеотекста [UNIVERSAL, 2l] с именем VideotexStringH, строки графических символов [UNIVERSAL,'25] с именем GraphicString.

Переменные типа NULL имеют единственное значение NULL и используются в тех случаях, если необходимо явно указать отсутствие параметра, например:

пациент : := SEQUENCE

{имя строка

номер палаты CHOICE

{INTEGER, NULL

--если пациент не находится в данный момент в больнице}}

До сих пор мы рассматривали простые типы. Абстрактный синтаксис ASN.1 позволяет структурировать заданные типы,

получая тем самым новые. Одним из способов структуризации является упорядочение уже имеющихся типов в виде последовательности. Такая упорядоченная последовательность определяется ключевым словом SEQUENCE и содержит перечисление типов с возможными указаниями OPTIONAL или DEFAULT после любого типа. Указание OPTIONAL означает, что значение данного типа в последовательности может быть опущено. Указание DEFAULT, за которым должно следовать «значение» данного типа, говорит о том, что если значение типа опущено, то по умолчанию восстанавливается указанное «значение». Кроме того, перед типом в списке может стоять приставка COMPONENT OF.

Это означает, что в список включается один из компонентов указанного типа. Когда последовательность состоит из значений одного и того же типа, для структуризации используется ключевое слово SEQUENCE OF. В этом случае использование OPTIONAL не разрешается.

Если упорядоченность не играет роли, используется структуризация множеством (SET). Как и в случае с упорядоченными последовательностями, здесь могут использоваться OPTIONAL и DEFAULT. Для множества из одинаковых типов вместо SET используется ключевое слово SET OF.

Примерами структуризации рассмотренными способами могут служить следующие описания:

```
Руководство : : = SEQUENCE {директор  строка
замдиректора OPTIONAL ISO646String. секретарь ISO646String};
Файл : := SEQUENCE блок;
Радуга : := SET
{красный, оранжевый, желтый, зеленый, голубой};
Почта : := SET OF почтовый ящик.
```

Выше было рассмотрено понятие тега. Тегируемый тип представляет собой новый тип, изоморфный тому, к которому применяется тегирование, но отличающийся от него тегом. В результате тегирования все значения нового типа будут отличаться от значений старого. Как указывалось выше, тег состоит из класса и номера  внутри класса, где

```
класс : := UNIVERSAL| APPLICATION |PRIVATE |ПУСТО
```



При передаче значения тегированного типа передается как тег, определенный пользователем, так и тег самого типа. В тех случаях, когда тег самого типа передавать не надо (однозначность разбора обеспечивается явным тегированием), можно использовать служебное слово IMPLICIT.

Примеры явного тегирования:

```
Клиент : := CHOICE
имя                 [0] IMPLICIT ISO6465String,
почтовыйадрес       [l] IMPLICIT ISO6465String,
учетныйномер        [2] единыйкод
единый код : := [PRIVATE, 10] IMPLICIT INTEGER
```

Для того чтобы представлять переменные, которые могут принимать одно из значений заданного набора переменных, служит конструкция CHOICE. Идентификация каждой такой переменной определяется контекстно-зависимым тегом (для класса альтернатива ПУСТО):

```
ИдентификаторФайла : :=
{относительноеИмя[0] IMPLICIT ISO646String
-- имя файла в справочнике,
абсолютноеИмя               [l] IMPLICIT ISO646String,
-- имя файла вместе с именем справочника,
номер                       [2] IMPLICIT INTEGER,
-- системный номер файла--}
```

Для выбора конкретных альтернатив типа CHOICE служит конструкция SELECT. Использование этой конструкции достаточно понятно из следующего примера.

Пусть есть описание:

```
Атрибутыфайла : := CHOICE
времяпоследнегоиспользования INTEGER,
имяфайла    ISO646String
```

Тогда текущие значения атрибутов

```
Текущиеатрибуты : := SEQUENCE
Времяпоследнегоиспользования      <Атрибутыфайла
имяфайла                          < Атрибутыфайла
```

Для обозначения значений, типы которых не специфицированы, используются служебные слова ANY (любой) и EXTERNAL (внешний). Первое обозначает допускающие значения любого типа, относящиеся к ASN.1, а второе указывает на значение, типы которых описаны иначе, чем средствами ASN.1, например:

Содержимоесообщения: := ANY;
Содержимоефайла : := EXTERNAL;
Списокдокументов : := SEQUENCE OF EXTERNAL.

Абстрактный синтаксис ASN.1 содержит механизм, который позволяет определять новые понятия и использовать их в дальнейшем для конструирования типов и значений. Это механизм макроопределений, описывающий синтаксис новых типов и значений, вводимых пользователем. Использование макроопределений сходно с использованием конструкции CHOICE, так как и в том, и другом случае тег всей конструкции не определен. Поэтому данные конструкции запрещено использовать там, где требуется заранее знать значение тега, кроме того, запрещено использовать неявное тегирование (IMPLICIT).

Для введения макроопределений используется служебное слово MACRO, а для определения новых типов и новых значений - ключевые слова TYPE NOTATION и VALUE NOTATION. Допустим, мы хотим расширить базовый набор ASN.1 макроопределением ПАРА 33, описывающим значения в виде пар (X=-------,Y=-------) где ------- значения любого типа ASN.1. Описание выглядит следующим образом:

```
ПАРА MACRO: : = BEGIN
 TYPE NOTATION: : =
            "ТИПХ""="type (тип 1)
            "ТИПУ" "=" type (тип 2)
VALUE NOTATION: : =
            "(" "X" "=" value (значение 1 тип 1)
            "," "Y""=" value (значение 2 тип 2)
< VALUE SEQUENCE   тип 1, тип 2   : :=
            {значение 1, значение 2} >
END.
```

Здесь type указывает, что каждый экземпляр нового понятия использует стандартные (или уже определенные) типы ASN.1 (тип 1,тип 2), а value связывает значения с этими типами. Результирующее значение макроопределения заключается в угловые скобки, а служебное слово VALUE используется в качестве локального имени, содержащего значение результирующего выражения.

Теперь, используя макроопределение ПАРА, можно определять новые типы, например, тип



```
Tl : := ПАРА        ТИПХ = INTEGER
                    ТИПУ = BOOLEAN,
```

одним из возможных значений которого может быть

T1 (X = 3, Y = TRUE).

Новый тип, в свою очередь, рекурсивно можно использовать в дальнейших определениях:

```
T2 : := ПАРА        ТИПХ = ISO646 String
                    ТИПУ = T1
```

Одним из возможных значений типа T2 будет

T2 (X = "Имя", Y = (X = 4, Y = FALSE)).

И, наконец, для структуризации описаний ASN.1, относящихся к одной предметной области, вводятся модули. Тело модуля охватывается скобками BEGIN и END и содержит определения ASN.1. Модуль имеет имя, после которого следует служебное слово DEFINITION. Имена модулей должны быть уникальными. Для международных стандартов рекомендуется использовать имена модулей в следующем виде:

ISOxxxx-yyyy,

где xxxx — номер международного стандарта, а yyyy — некоторое сокращение, используемое для этого стандарта. Пример описания модуля для стандарта передачи, доступа и манипулирования файлом (FTAM):

```
ISO8571 = FTAM DEFINITION : : =
BEGIN
  PDU : —CHOICE {initializePDU, FilePDU, BulcdataPDU}
END.
```

Доступ к определениям модуля осуществляется с использованием точечной нотации:

```
A : := ISO8571 = FTAM.PDU
```

*Базовые правила кодирования для ASN.1*

В принципе могут существовать различные наборы правил для кодирования ASN.1, формирующих синтаксисы передачи. Среди множества таких правил выделяются базовые правила кодирования ASN.1, принятые в настоящее время в качестве стандарта.

В соответствии с этими правилами кодирование всех типов, за исключением внешних, могут быть представлены в двух видах. В одном из них размер содержимого указывается явно,

в другом — определяется по признаку конца содержимого (два нулевых октета). В дальнейшем считается, что биты октетов нумеруются справа налево от 1 до 8, где бит 8 — старший значащий бит. Общие формы кодирования с явным указанием длины и с признаком конца содержимого имеют следующий вид (рис. 3.34):

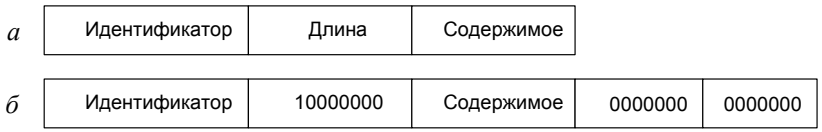



Рисунок 3.34 — Общие формы кодирования:
*а* — с явным указанием длины, *б* — с признаком конца содержимого

Идентификатор представляет собой тег (класс и номер) типа значения, как это определено выше. Комбинация значений бит 7 и 8 в первом октете идентификатора определяет класс следующим образом: 00 — UNIVERSAL, 01 — контекстно-зависимый, 10 — APPLICATION, 11 — PRIVATE. Ноль в бите 6 указывает на то, что данный тип простой, а единица — сложный. Если значение номера в теге меньше 31, то это значение размещается в первых пяти битах и идентификатор в этом случае состоит из одного октета. Если значение номера в теге больше 30, то первые пять бит содержат единицы, а значение размещается в битах последующих октетов, причем бит 8 последнего октета содержит ноль, а в промежуточных октетах он устанавливается в единицу.

Длина содержимого, если она меньше 128, состоит из одного октета, бит 8 в котором установлен в нуль, а биты 7...1 указывают длину содержимого. Если длина больше 127, то используется прием, рассмотренный раннее. В этом случае значение длины располагается в семи младших разрядах последовательности октетов, последний из которых содержит нуль в старшем бите (бит 8), а остальные — единицы в этом бите.



Поле «содержимое» состоит из последовательности октетов (начиная с нулевого), которые представляют значения данных.

Значения булевских типов (BOOLEAN) представляются одним октетом. При этом значение FALSE состоит из октета, содержащего все нули, а значение TRUE представляется октетом с любым (на усмотрение реализатора) ненулевым значением.

Значения целого (INTEGER) может состоять из одного или более октетов. Октеты содержимого в этом случае представляют значение целого в двоичном дополнительном коде, составленное в последовательности биты 8…1 первого октета, затем биты 8...1 второго октета и т.д., вплоть до последнего октета поля «Содержимое».

Пустое значение (NULL) представляется следующим образом: поле длины заполняется нулями, а содержимое не содержит ни одного октета.

Для представления значений применяется простое или сложное кодирование. При простом кодировании октеты содержимого непосредственно представляют кодируемое значение. Этот метод используется для кодирования значений простых типов BOOLEAN,INTEGER и NULL. Сложное кодирование заключается в том, что кодируемое значение представляется одним или несколькими значениями октетов содержимого. Этот вид кодирования применяется для представления значений структурированных типов. Значения типов BITSTRING и OCTETSTRING могут представляться по усмотрению разработчика любым из описанных методов. Рекомендуется использовать метод сложного кодирования в тех случаях, когда передается часть строки, которая еще полностью не сформирована.

В случае простого кодирования значений типов BITSTRING и OCTETSTRING содержимое непосредственно представляет собой последовательность бит или октетов соответственно, причем в случае BITSTRING последовательность дополняется до кратного числа октетов и начинается специальным октетом, указывающим количество значащих бит в последнем октете. Биты в строках BITSTRING располагаются следующим образом: за специальным октетом, указывающим число

значащих бит последнего октета, идут биты 8...1 первого октета строки, затем биты 8...1 второго октета и т.д. В пустой строке типа OCTETSTRING поле «Содержимое» отсутствует, а пустая строка типа BITSTRING содержит один октет с нулевым значением.

В случае сложного кодирования битовых строк или строк октетов каждое значение, если таковые имеются, содержат полное представление данного типа, включающее идентификатор, длину, октеты содержимого и, возможно, признак конца содержимого. В частности, тег значений типа BITSTRING всегда будет состоять из класса UNIVERSAL и номера 3, а типа OCTETSTRING — класса UNIGERSAL и номера 4.

Рассмотрим, как представляются структурированные значения с использованием сложного кодирования. Поля «Содержимое» для структурированных типов SEQUENCE, SEQUENCE OF, SET, SET OF состоят из последовательности представлений данных для каждого типа, заданного в описании на ASN.1, причем порядок их представления для SEQUENCE и SEQUENCE OF совпадает с порядком их вхождения в описаниях на ASN.1, а для SET и SET OF — произволен. Для SEQUENCE и SET представления составляющих их данных, типы которых содержат приставки OPTIONAL и DEFAULT, могут быть опущены. Представление значений типов CHOICE и SELECTION будет совпадать с представлением соответственно выбранного или селектированного типа в конкретной конструкции. Приставка IMPLICIT в описании типа на ASN.1 устраняет необходимость передавать тег значения типа, следующего после IMPLICIT.

В качестве примера использования ASN.1 и базовых правил кодирования рассмотрим гипотетические записи о сотрудниках предприятия. Пусть такие записи должны представлять множества людей с указанием для каждого из этих людей имени, должности, табельного номера, даты поступления на работу, имени жены, числа детей и информацию о детях, включающую имена детей и даты их рождения. Описание таких записей в виде абстрактного синтаксиса ASN.1 выглядит следующим образом:



```
Записьсотрудников : := [APPLICATION O] IMPLICIT SET
                       ФИО
  должность            IS0646String,
  номер                табельныйномер
  датапоступления      [1]  дата
  имяжены              [2] ФИО
  дети                 [3] IMPLICIT SEQUENCE OF
                       информацияодетях
Информацияодетях       ::= SET
{                      ФИО,
  датарождения         [0] дата }

ФИО : := [APPLICATION 1] IMPLICIT SEQUENCE
{ фамилия        ISO646String,
  имя            ISO646String,
  отчество       ISO646String}

Табельныйномер : := [APPLICATION 2] IMPLICIT INTEGER
Дата : := [APPLICATION 3] IMPLICIT ISO646String
```

Пусть, используя это описание, мы хотим передать следующую запись, состоящую из сведений об одном сотруднике:

Запись:

| | |
|---|---|
| Фамилия, имя, отчество | : Сидоров Иван Петрович |
| Должность | : инженер |
| Табельный номер | : 100 |
| Дата поступления | : 4 марта 1970 |
| Фамилия, имя, отчество жены | : Сидорова Татьяна Николаевна |
| Число детей | : 2 |
| Информация о детях | |
| Фамилия, имя, отчество | : Сидоров Владимир Иванович |
| Дата рождения | : 10 июля 1987 года |
| Информация о детях | |
| Фамилия, имя, отчество | : Сидорова Валентина Ивановна |
| Дата рождения | : 8 августа 1965 |

Используем фигурные скобки для того, чтобы объединять элементы записи в группы. Тогда структура этой записи может быть представлена следующим образом:

{{фамилия "Сидоров", имя "Иван", отчество "Петрович"},
должность "инженер"
табельныйномер 100
датапоступления "19700304"
имяжены {фамилия "Сидорова", имя "Татьяна",
отчество "Николаевна" },
дети
{{{фамилия "Сидоров", имя "Владимир",
отчество "Иванович"},
датарождения "19870710" },
{{фамилия "Сидорова", имя "Валентина",
отчество "Ивановна"}
датарождения "19650808" }}}

Применение к этой записи базовых правил кодирования позволяет закодировать ее в информационный поток так (рис. 3.35, 3.36), что при приеме данной информации запись будет полностью восстановлена. Значения идентификаторов, длин и целых представлены в 16-ричном виде двумя цифрами в октете.

Следует обратить внимание на значение общей длины содержимого записи сотрудников. Так как общая длина содержимого составляет 180 (в десятичном виде) октетов, что больше 127, то используется метод расширения, описанный выше.



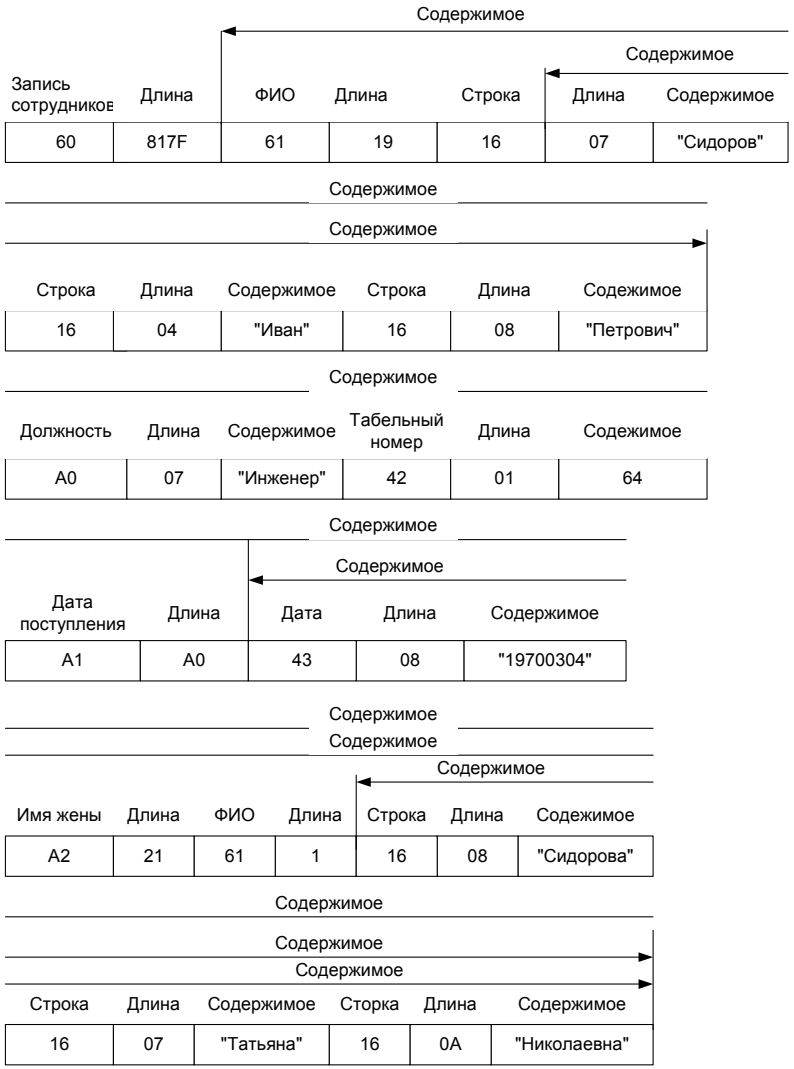



Рисунок 3.35 — Представление закодированной записи

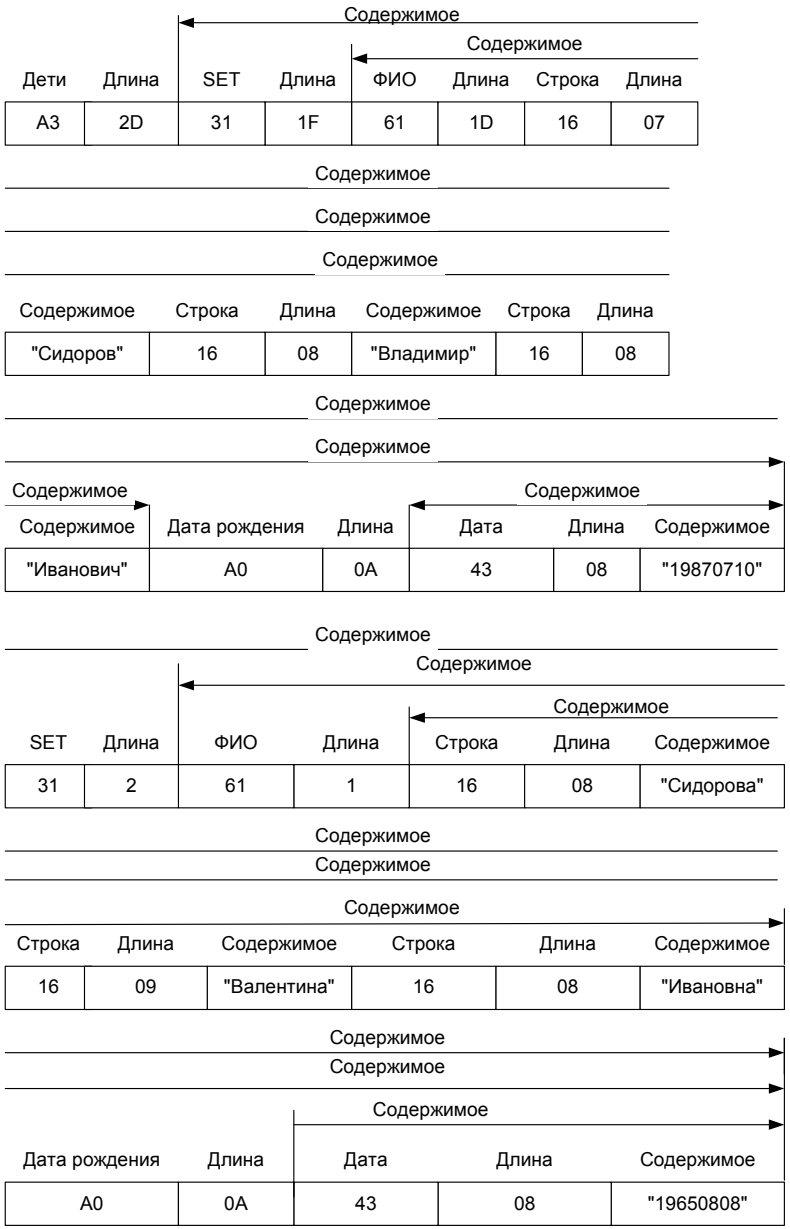



Рисунок 3.36 — Представление закодированной записи



## 3.12 Общий прикладной сервис

Целью выделения общего прикладного сервиса в отдельный стандарт является обеспечение разработчиков прикладных распределенных систем обработки информации наиболее общими унифицированными механизмами — прикладными услугами, которые в совокупности с услугами представительного сервиса составляют понятийную основу построения и работы приложений.

Спецификация последовательностей отработки общих прикладных услуг обеспечивает, со своей стороны, метод организации взаимодействия прикладных систем. В этом заключается большое гносеологическое и методологическое значение стандарта. Стандарт освещает два основных аспекта — работа ориентированных на соединение приложений, образование ассоциаций взаимодействующих сторон и функционирование приложений на фоне сбоев и отказов компонентов распределенной системы, в частности, обеспечение свойства неделимости выполняемых распределенных действий. Прикладной уровень является последним седьмым уровнем модели ВОС. С этим связан ряд особенностей, отличающих его от других уровней эталонной модели. На прикладном уровне завершается «вертикальное» наращивание сервисной мощности модели ВОС и начинается «горизонтальное». Сервис, поставляемый различными протокольными прикладными объектами с учетом определенных ограничений, можно объединять, формируя различные профили сервиса прикладного уровня в нуждах конкретных, специальных прикладных систем.

На прикладном уровне выполняется окончательное сращивание модели ВОС с конкретными вычислительными операционными системами, осуществляется вживление механизмов взаимосвязи, объявленных в модели, в вычислительную среду с ее понятийным построением. В связи с этим среди прочих категорий прикладного уровня в стандарте уточняется понятие прикладного процесса Application process (AP). Прикладной процесс — это идентифицируемый объект в рамках реальной открытой системы, ведущий обработку информации и ответственный за согласование соглашений среды своего существования с законами модели ВОС. Прикладной процесс представля-

ется в рамках модели ВОС одним и более прикладными объектами. С этих позиций и следует рассматривать стандарт, специфицирующий общие прикладные услуги, которые по усмотрению разработчиков специальных распределенных систем могут быть использованы или не использованы, включены или не включены в профиль применяемого для построения приложения прикладного сервиса.

Ниже приводится описание двух компонентов, стандартизуемых в документах.

С позиций модели ВОС объекты, которые формируют прикладное взаимодействие, являются прикладными объектами — Application Entity (AE). Схема взаимодействия прикладных объектов предполагает их распределение на ведущие и ведомые с точки зрения инициализации действия.

Прикладной объект представляется множеством прикладных услуг — Application Service Elements (ASE), которое, в свою очередь, содержит набор общих прикладных услуг (ACSE), некоторый набор специальных прикладных услуг (SASE), например услуги передачи и управления файлами (FTAM) и/или управления передачей и выполнением заданий (JTM), а также услуги, создаваемые разработчиком прикладной системы — услуги пользователя (UE), то есть

ASE = {ACSE, SASE, UE}.

Множество прикладных услуг, таким образом, задает сервисный профиль прикладного элемента, обеспечивающего функционирование специального приложения. В процессе отработки прикладной услуги могут использоваться другие услуги того АЕ, которому она принадлежит, или же сервис представительного уровня. В то же время прикладная услуга может применяться при реализации другой прикладной услуги или же непосредственно услуги пользователя, принадлежащей тому же прикладному объекту. В наборе общих прикладных услуг выделяется подмножество, называемое базовым ядром ACSE. Базовое ядро ACSE обеспечивает:

- поддержку общих схем именования и адресации прикладных объектов, отображение имен прикладных объектов в соответствующие адреса представительных сервисных точек доступа;



- согласование прикладного контекста взаимодействия в рамках приложения.

Кроме того, базовое ядро ACSE предоставляет стандартный метод установления связи между прикладными объектами, который позволяет отклонить запрос как на установление, так и на завершение прикладного соединения со стороны пользователя; организовать упорядоченное (нормальное) или безусловное (ненормальное) завершение прикладного соединения.

Прикладное взаимодействие базируется на представительном соединении, которое создается и завершается поставщиком общего прикладного сервиса. Поэтому пользователь сервиса не должен устанавливать представительное соединение прямым выходом на представительный уровень с помощью примитива P-CONNECT. Доступ к базовому ядру ACSE, обеспечивающему услугу образования ассоциации (прикладного соединения), может быть осуществлен опосредованно вызовом специальной прикладной услуги или услуги пользователя. Прикладное взаимодействие, развивающееся на установленном прикладном соединении, характеризуется множеством используемых в его ходе общих прикладных услуг и множеством специальных прикладных услуг. Эти множества формируют так называемый прикладной контекст ассоциации.

В соответствии со стандартом базовое ядро ACSE предназначено для управления прикладным взаимодействием и содержит следующие прикладные услуги: A-ASSOCIATE, A-RELEASE, A-U-ABORT и A-P-ABORT. Услуга A-ASSOCIATE позволяет прикладному объекту устанавливать прикладное соединение (ассоциацию) с другим прикладным объектом. В ходе установления ассоциации прикладные объекты обмениваются параметрами прикладного соединения и согласуют их, в частности согласуется прикладной контекст ассоциации. Услуга A-RELEASE позволяет прикладному объекту произвести упорядоченное завершение существующего прикладного соединения без потери передаваемой информации. Услуга A-U-ABORT используется прикладным объектом для безусловного завершения прикладного соединения. С помощью услуги A-P-ABORT поставщик общего прикладного сервиса информирует прикладной объект о выполненном им безусловном завершении прикладного соединения.

### 3.12.1 Сервис обеспечения неделимости распределенных действий

Одновременное функционирование прикладных распределенных систем выдвигает ряд общих требований, которые необходимо учитывать в ходе их разработки и поддерживать в реализациях. Эти требования связаны с обеспечением соответствия поведения реализации прикладного распределенного приложения его функциональной спецификации независимо от сбоев и отказов, как открытых систем носителей приложения, так и средств связи. Помимо этого, параллельная работа различных приложений не должна сказываться на результатах деятельности каждого отдельно взятого приложения (приложения не должны интерферировать), и, наконец, зачастую действие, развивающееся в рамках распределенного приложения, должно обладать свойством неделимости. В документах определяется понятийная и методологическая основа, позволяющая удовлетворить указанным требованиям. В них рассматриваются три аспекта, объединенных под общим названием «Commitment, Concurrency and Recovery») (CCR), фиксация, параллельность и восстановление в отношении организации распределенного действия.

Распределенное приложение охватывает ряд (не менее двух) открытых систем. Каждая открытая система содержит элемент-приложение. Распределенное действие, выполняемое приложением, заключается в целенаправленном взаимодействии элемент-приложений. В ходе взаимодействия элемент-приложения образуют и контролируют восстанавливаемые данные, вступая попарно между собой в CCR-отношения, которые сохраняются на фоне отказов открытых систем и связи, устанавливаются и завершаются с помощью специальных CCR-примитивов в рамках существующих ассоциаций. Действия элемент-приложений направлены на обработку целевых (граничных) данных (Bound Data), которые являются восстанавливаемыми данными, и характеризуются состоянием, зависящим от фазы CCR-отношения.

Обеспечение свойства неделимости распределенного действия требует от элемент-приложения поддержания восстанавливаемых данных, называемых *данными неделимого действия,*



которые сохраняют состояние CCR-отношения, а от системы носителя элемент-приложения — наличие *механизма управления восстановлением,* гарантирующего корректное развитие неделимого действия на фоне отказов. Под неделимым действием понимается последовательность операций, выполняемых распределенным приложением, обладающая следующими свойствами:

- управляется (прямо или косвенно) единственным элемент-приложением;

- развивается без интерференции с внешними действиями;

- части неделимого действия исполняются различными элемент - приложениями и либо

а) завершаются успешно *(фиксация* — завершение неделимого действия, сопровождаемое переводом целевых данных в конечное состояние и обеспечением их доступности другим действиям, а также расторжением CCR-отношений),

б) целевые данные остаются без изменения, причем управляющий элемент-приложение получает одно или более диагностических сообщений, поставляемых элемент-приложениями, вовлеченными в неделимое действие (возврат — завершение неделимого действия, сопровождаемое переводом целевых данных в исходное состояние, а также расторжением CCR-отношения).

Организация неделимого действия подразумевает наличие механизма *управления параллельностью* доступа к целевым данным, который гарантирует, что неделимое действие не завершится до тех пор, пока:

а) не завершатся все неделимые действия, которые обрабатывали те же целевые данные до их периода использования рассматриваемым неделимым действием;

б) никаких изменений значений целевых данных в течение их периода использования не произойдет, за исключением тех, которые планируются данным неделимым действием.

Механизм управления параллельностью может базироваться на технике блокирования используемых ресурсов. Однако возможно применение других алгоритмов управления параллельностью, связанных с менее строгим упорядочением неделимых действий, допущением большего числа случаев возврата.

Выбор техники управление параллельностью остается за реализаторами прикладных распределенных систем.

Установление CCR-отношения подразумевает разделение его участников на ведущего (элемент-приложение, инициировавшее CCR-отношение) и ведомого (элемент-приложение, принявшее CCR-отношение). Неделимое действие, вовлекая в свой ход множество элемент-приложений, образует дерево неделимого действия, ветвями которого являются CCR-отношения, развивающиеся на существующих двунаправленных ассоциациях. Таким образом, неделимое действие охватывает множество ведущих и ведомых, причем среди ведущих существует источник неделимого действия — корень неделимого действия. Каждая ветвь неделимого дерева — CCR-отношение — имеет свою уникальную для данного неделимого действия идентификацию. В ходе развития неделимого действия одно и то же элемент-приложение может задействоваться неоднократно, то есть иметь несколько вхождений в дерево неделимого действия, причем в различных качествах (рис. 3.37).

| Ведущие | Ведомые |
|---|---|
| A | B, C |
| B | D |
| C | E, F, G |
| E | F, G |
| G | J |

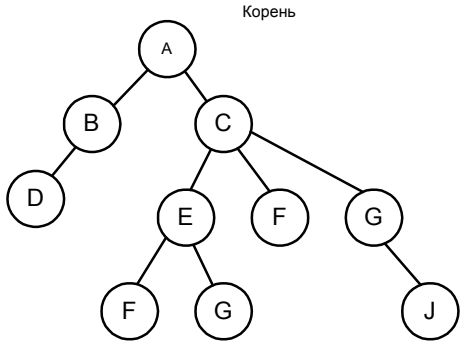



Рисунок 3.37 — Дерево неделимого действия

Придание свойства неделимости некоторому действию достигается с помощью выполнения этого действия в две фазы. В первой фазе источник действия определяет, все ли ведомые способны выполнить его фиксацию. И только после получения отчетов от всех ведомых о степени готовности к выполнению



фиксации действия принимается решение о том, выполнять ли фиксацию. В ходе второй фазы ведомые осуществляют либо фиксацию, либо возврат в соответствии с принятым решением (рис. 3.38). Эта схема обеспечения свойства неделимости действует при соблюдении следующих условий:

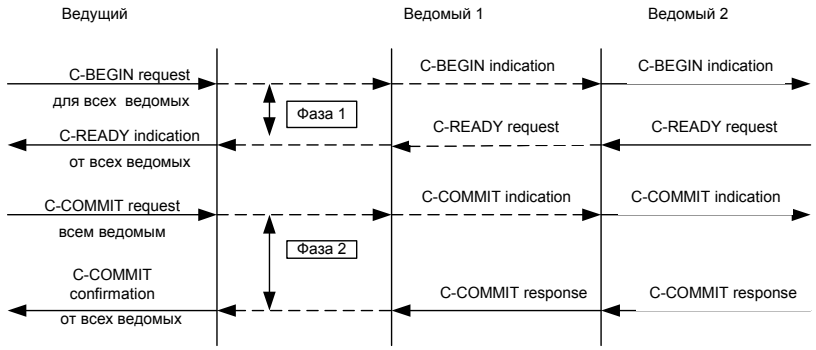
- в фазе фиксации результатов неделимого действия не должно существовать возможностей их изменения;
- ведущий может потребовать выполнения возврата к начальному состоянию в любое время до выдачи запроса на фиксацию результатов;
- ведущий не может требовать выполнения фиксации результатов от ведомого до тех пор, пока не получит от него согласия на фиксацию;
- если ведомый сообщил о своей готовности выполнить фиксацию, то он не может отказаться от требования выполнить ее;
- ведомый может отказаться от фиксации в любое время вплоть до момента выдачи своего согласия на фиксацию;
- ведущий требует выполнить возврат к исходному состоянию от ведомого, указавшего на невозможность фиксации результатов.

Корень неделимого действия является конечным приемником диагностики, выдаваемой ведомыми. Корень несет ответственность за выполнение фиксации или возврата. Имя корня используется для идентификации неделимого действия в целях управления параллельностью. Природа неделимого действия определяется прикладным протоколом, в соответствии с которым осуществляется работа в рамках неделимого действия от момента его начала и до момента выполнения фиксации.

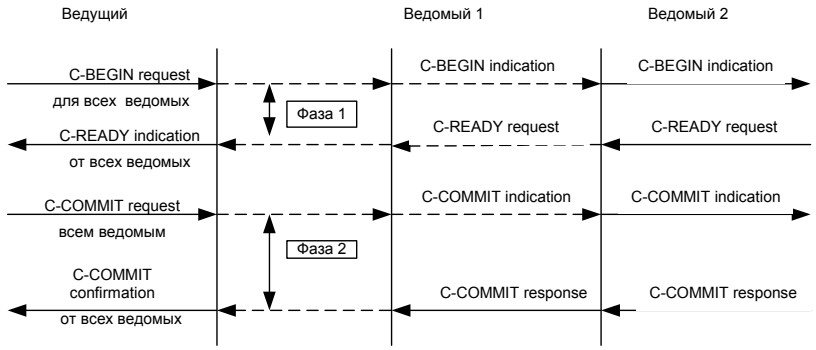
Характер неделимого действия может меняться в зависимости от того, отказались ли некоторые или все ведомые от выполнения фиксации. Завершение неделимого действия вызывает высвобождение ресурсов, затрачиваемых им, то есть целевых данных, данных неделимого действия, относящихся к управлению параллельностью. Уничтожаются все данные сопровождения неделимого действия.

*Услуги фиксации, параллельности и восстановления (CCR).* Услуга инициализации неделимого действия C-BEGIN обеспечивает установление CCR-отношения (рис. 3.38). В результате

дерево неделимого действия приобретает еще одну ветвь. Услуга с точки зрения CCR является неподтверждаемой. Отказ партнера образовать CCR-отношение выполняется с помощью услуги C-REFUSE. Действие услуги C-BEGIN связано с установлением точки большой синхронизации в рамках используемой ассоциации. Вслед за услугой C-BEGIN прикладной пользователь может обращаться к другим прикладным услугам, что, собственно, и определяет сущность вводимого неделимого действия.




*а*




*б*

Рисунок 3.38 — Две фазы выполнения неделимого действия:
*а* — фиксация; *б* — возврат



# 4 СТЕКИ ПРОТОКОЛОВ И ПРОТОКОЛЫ СЕТЕЙ ЭВМ

## 4.1 Обзор существующих стеков протоколов

Важнейшим направлением стандартизации в области вычислительных сетей является стандартизация коммуникационных протоколов. В настоящее время в сетях используется большое количество стеков коммуникационных протоколов. Наиболее популярными являются стеки: TCP/IP, IPX/SPX, NetBIOS/SMB, DECnet, SNA и OSI. Все эти стеки, кроме SNA, на нижних уровнях — физическом и канальном, — используют одни и те же, хорошо стандартизованные, протоколы Ethernet, Token Ring, FDDI и некоторые другие, которые позволяют использовать во всех сетях одну и ту же аппаратуру. Зато на верхних уровнях все стеки работают по своим собственным протоколам. Эти протоколы часто не соответствуют рекомендуемому моделью OSI разбиению на уровни. В частности, функции сеансового и представительного уровня, как правило, объединены с прикладным уровнем. Такое несоответствие связано с тем, что модель OSI появилась как результат обобщения уже существующих и реально используемых стеков, а не наоборот.

*Стек OSI*

Следует четко различать модель OSI и стек OSI. В то время как модель OSI является концептуальной схемой взаимодействия открытых систем, стек OSI представляет собой набор вполне конкретных спецификаций протоколов. В отличие от других стеков протоколов стек OSI полностью соответствует модели OSI, он включает спецификации протоколов для всех семи уровней взаимодействия, определенных в этой модели. На нижних уровнях стек OSI поддерживает Ethernet, Token Ring, FDDI, протоколы глобальных сетей, X.25 и ISDN, — то есть использует разработанные вне стека протоколы нижних уровней, как и все другие стеки. Протоколы сетевого, транспортного и сеансового уровней стека OSI специфицированы и реализованы различными производителями, но распространены пока мало. Наиболее популярными протоколами стека OSI являются прикладные протоколы. К ним относятся протокол передачи файлов

FTAM, протокол эмуляции терминала VTP, протоколы справочной службы X.500, электронной почты X.400 и ряд других. Протоколы стека OSI отличает большая сложность и неоднозначность спецификаций. Эти свойства явились результатом общей политики разработчиков стека, стремившихся учесть в своих протоколах все случаи жизни и все существующие и появляющиеся технологии. К этому нужно еще добавить и последствия большого количества политических компромиссов, неизбежных при принятии международных стандартов по такому злободневному вопросу, как построение открытых вычислительных сетей. Из-за своей сложности протоколы OSI требуют больших затрат вычислительной мощности центрального процессора, что делает их наиболее подходящими для мощных машин, а не для сетей персональных компьютеров. Стек OSI — международный, независимый от производителей стандарт. Его поддерживает правительство США в своей программе GOSIP, в соответствии с которой все компьютерные сети, устанавливаемые в правительственных учреждениях США после 1990 года, должны или непосредственно поддерживать стек OSI, или обеспечивать средства для перехода на этот стек в будущем. Тем не менее стек OSI более популярен в Европе, чем в США, так как в Европе осталось меньше старых сетей, работающих по своим собственным протоколам. Большинство организаций пока только планируют переход к стеку OSI, и очень немногие приступили к созданию пилотных проектов. Из тех, кто работает в этом направлении, можно назвать Военно-морское ведомство США и сеть NFSNET. Одним из крупнейших производителей, поддерживающих OSI, является компания AT&T, ее сеть Stargroup полностью базируется на этом стеке.

*Стек TCP/IP*

Стек TCP/IP был разработан по инициативе Министерства обороны США в 60-70-х годах XX века, для связи экспериментальной сети ARPAnet с другими сетями как набор общих протоколов для разнородной вычислительной среды. Большой вклад в развитие стека TCP/IP внес университет Беркли, реализовав протоколы стека в своей версии ОС UNIX. Популярность этой операционной системы привела к широкому распространению протоколов TCP, IP и других протоколов стека. Сегодня этот стек используется для связи компьютеров всемирной информационной сети Internet, а также в огромном числе корпоративных сетей. Стек TCP/IP на нижнем уровне поддерживает все популярные стандарты физического и канального уровней: для локальных сетей — это Ethernet, Token Ring, FDDI, для глобальных — протоколы работы на аналоговых коммутируемых и выделенных линиях SLIP, PPP, протоколы территориальных сетей X.25 и ISDN. Основными протоколами стека, давшими ему название, являются протоколы IP и TCP. Эти протоколы в терминологии модели OSI относятся к сетевому и транспортному уровням соответственно. IP обеспечивает продвижение пакета по составной сети, а TCP гарантирует надежность его доставки. За долгие годы использования в сетях различных стран и организаций стек TCP/IP вобрал в себя большое количество протоколов прикладного уровня. К ним относятся такие популярные протоколы, как протокол пересылки файлов FTP, протокол эмуляции терминала telnet, почтовый протокол SMTP, используемый в электронной почте сети Internet, гипертекстовые сервисы службы WWW и многие другие. Сегодня стек TCP/IP представляет собой один из самых распространенных стеков протоколов вычислительных сетей. Действительно, только в сети Internet объединено около 10 миллионов компьютеров по всему миру, которые взаимодействуют друг с другом с помощью стека протоколов TCP/IP. Стремительный рост популярности Internet привел и к изменениям в расстановке сил в мире коммуникационных протоколов — протоколы TCP/IP, на которых построен Internet, стали быстро теснить бесспорного лидера прошлых лет — стек IPX/SPX компании Novell. Сегодня в мире общее количество компьютеров, на которых установлен стек TCP/IP, сравнялось с общим количеством компьютеров, на которых работает стек IPX/SPX, и это говорит о резком переломе в отношении администраторов локальных сетей к протоколам, используемым на настольных компьютерах, так как именно они составляют подавляющее число мирового компьютерного парка и именно на них раньше почти везде работали протоколы компании Novell, необходимые для доступа к файловым серверам NetWare. Процесс становления стека TCP/IP в качестве стека номер один в любых типах сетей

продолжается, и сейчас любая промышленная операционная система обязательно включает программную реализацию этого стека в своём комплекте поставки. Хотя протоколы TCP/IP неразрывно связаны с Internet и каждый из многомиллионной армады компьютеров Internet работает на основе этого стека, существует большое количество локальных, корпоративных и территориальных сетей, непосредственно не являющихся частями Internet, в которых также используют протоколы TCP/IP. Чтобы отличать их от Internet, эти сети называют сетями TCP/IP или просто IP-сетями. Поскольку стек TCP/IP изначально создавался для глобальной сети Internet, он имеет много особенностей, дающих ему преимущество перед другими протоколами, когда речь заходит о построении сетей, включающих глобальные связи. В частности, очень полезным свойством, делающим возможным применение этого протокола в больших сетях, является его способность фрагментировать пакеты. Действительно, большая составная сеть часто состоит из сетей, построенных на совершенно разных принципах. В каждой из этих сетей может быть установлена собственная величина максимальной длины единицы передаваемых данных (кадра). В таком случае при переходе из одной сети, имеющей большую максимальную длину, в сеть с меньшей максимальной длиной может возникнуть необходимость деления передаваемого кадра на несколько частей. Протокол IP стека TCP/IP эффективно решает эту задачу. Другой особенностью технологии TCP/IP является гибкая система адресации, позволяющая более просто по сравнению с другими протоколами аналогичного назначения включать в интерсеть сети других технологий. Это свойство также способствует применению стека TCP/IP для построения больших гетерогенных сетей. В стеке TCP/IP очень экономно используются возможности широковещательных рассылок. Это свойство совершенно необходимо при работе на медленных каналах связи, характерных для территориальных сетей. Однако, как и всегда, за получаемые преимущества надо платить, и платой здесь оказываются высокие требования к ресурсам и сложность администрирования IP-сетей. Мощные функциональные возможности протоколов стека TCP/IP требуют для своей реализации высоких вычислительных затрат. Гибкая система адресации и отказ от широковещательных рассылок приводят к наличию в IP-сети различных централизованных служб типа DNS, DHCP и т. п. Каждая из этих служб направлена на облегчение администрирования сети, в том числе и на облегчение конфигурирования оборудования, но в то же время сама требует пристального внимания со стороны администраторов. Можно приводить и другие доводы за и против стека протоколов Internet, однако факт остается фактом — сегодня это самый популярный стек протоколов, широко используемый как в глобальных, так и локальных сетях.



*Стек IPX/SPX*

Этот стек является оригинальным стеком протоколов фирмы Novell, разработанным для сетевой операционной системы NetWare еще в начале 80-х годов. Протоколы сетевого и сеансового уровней Internetwork Packet Exchange (IPX) и Sequenced Packet Exchange (SPX), которые дали название стеку, являются прямой адаптацией протоколов XNS фирмы Xerox, распространенных в гораздо меньшей степени, чем стек IPX/SPX. Популярность стека IPX/SPX непосредственно связана с операционной системой Novell NetWare, которая еще сохраняет мировое лидерство по числу установленных систем, хотя в последнее время ее популярность несколько снизилась и по темпам роста она отстает от Microsoft Windows NT. Многие особенности стека IPX/SPX обусловлены ориентацией ранних версий ОС NetWare (до версии 4.0) на работу в локальных сетях небольших размеров, состоящих из персональных компьютеров со скромными ресурсами. Понятно, что для таких компьютеров компании Novell нужны были протоколы, на реализацию которых требовалось бы минимальное количество оперативной памяти (ограниченной в IBM-совместимых компьютерах под управлением MS-DOS объемом 640 Кбайт) и которые бы быстро работали на процессорах небольшой вычислительной мощности. В результате протоколы стека IPX/SPX до недавнего времени хорошо работали в локальных сетях и не очень — в больших корпоративных сетях, так как они слишком перегружали медленные глобальные связи широковещательными пакетами, которые интенсивно используются несколькими протоколами этого стека (например, для установления связи между клиентами и сервера-

ми). Это обстоятельство, а также тот факт, что стек IPX/SPX является собственностью фирмы Novell и на его реализацию нужно получать лицензию (то есть открытые спецификации не поддерживались), долгое время ограничивали распространенность его только сетями NetWare. Однако с момента выпуска версии NetWare 4.0 Novell внесла и продолжает вносить в свои протоколы серьезные изменения, направленные на их адаптацию для работы в корпоративных сетях. Сейчас стек IPX/ SPX реализован не только в NetWare, но и в нескольких других популярных сетевых ОС, например: SCO UNIX, Sun Solaris, Microsoft Windows NT.

*Стек NetBIOS/SMB*

Этот стек широко используется в продуктах компаний IBM и Microsoft. На физическом и канальном уровнях этого стека используются все наиболее распространенные протоколы Ethernet, Token Ring, FDDI и другие. На верхних уровнях работают протоколы NetBEUI и SMB. Протокол NetBIOS (Network Basic Input/Output System) появился в 1984 году как сетевое расширение стандартных функций базовой системы ввода/вывода (BIOS) IBM PC для сетевой программы PC Network фирмы IBM. В дальнейшем этот протокол был заменен так называемым протоколом расширенного пользовательского интерфейса NetBEUI — NetBIOS Extended User Interface. Для обеспечения совместимости приложений в качестве интерфейса к протоколу NetBEUI был сохранен интерфейс NetBIOS. Протокол NetBEUI разрабатывался как эффективный протокол, потребляющий немного ресурсов и предназначенный для сетей, насчитывающих не более 200 рабочих станций. Этот протокол содержит много полезных сетевых функций, которые можно отнести к сетевому, транспортному и сеансовому уровням модели OSI, однако с его помощью невозможна маршрутизация пакетов. Это ограничивает применение протокола NetBEUI локальными сетями, не разделенными на подсети, и делает невозможным его использование в составных сетях. Некоторые ограничения NetBEUI снимаются реализацией этого протокола NBF (NetBEUI Frame), которая включена в операционную систему Microsoft Windows NT. Протокол SMB (Server Message Block)



выполняет функции сеансового, представительного и прикладного уровней. На основе SMB реализуется файловая служба, а также службы печати и передачи сообщений между приложениями. Стеки протоколов SNA фирмы IBM, DECnet корпорации Digital Equipment и AppleTalk/AFP фирмы Apple применяются в основном в операционных системах и сетевом оборудовании этих фирм. Часто это соответствие весьма условно, так как модель OSI — это только руководство к действию, причем достаточно общее, а конкретные протоколы разрабатывались для решения специфических задач, причем многие из них появились до разработки модели OSI. В большинстве случаев разработчики стеков отдавали предпочтение скорости работы сети в ущерб модульности — ни один стек, кроме стека OSI, не разбит на семь уровней. Чаще всего в стеке, например TCP/IP, явно выделяются 3–4 уровня: уровень сетевых адаптеров, в котором реализуются протоколы физического и канального уровней, сетевой уровень, транспортный уровень и уровень служб, вбирающий в себя функции сеансового, представительного и прикладного уровней (рис. 4.1).

## 4.2 Стек TCP/IP

Этот стек называют также Интернет-протоколом. Основа стека разработана в 1970-х годах оборонным ведомством США (DoD) и различными исследовательскими организациями. Сегодня это один из наиболее популярных стеков протоколов, использующийся как в локальных сетях, так и для объединения гетерогенных сетей в единую глобальную сеть.

Интернет протокол определяет функции, соответствующие уровням модели ВОС выше канального уровня. Низкоуровневые протоколы умышленно оставлены открытыми для свободной реализации, для того чтобы в будущем TCP/IP можно было надстраивать над любой базовой технологией передачи данных.

Протокол эмуляции терминала (TELNET), протокол передачи файлов (FTP), простой протокол передачи почты (SMTP), протокол передачи гипертекста (HTTP), протокол сетевой файловой системы (NFS) и другие протоколы, соответствующие прикладному уровню, уровню представления и сеансовому

уровню называются процесс-протоколами. Процесс-протокол обеспечивает какой-либо конкретный сервис прикладного уровня, то есть позволяет решать именно те задачи, ради которых и создаются вычислительные сети. Эти вопросы обсуждались в главе 2 «Архитектура и стандарты сетей ЭВМ».

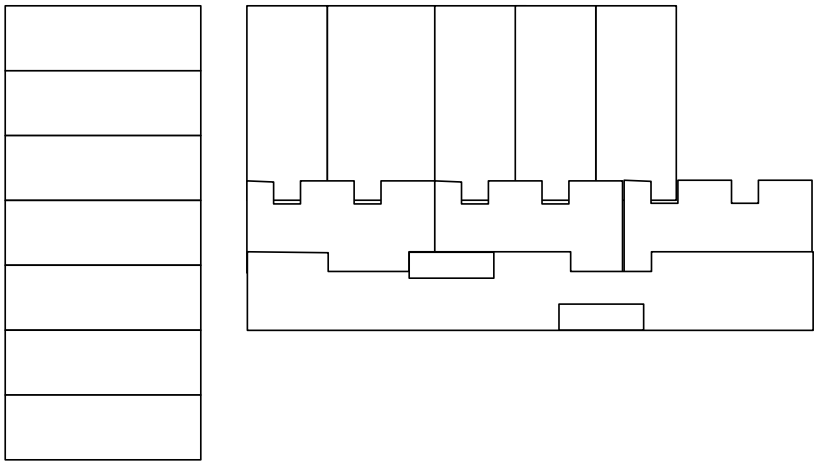


Рисунок 4.1 — Состав стека TCP/IP и соответствие модулей уровням модели ВОС

Протокол контроля передачи (TCP) и протокол пользовательских датаграмм (UDP) — являются так называемыми протоколами типа хост-хост (host-to-host). Хост-хост протоколы решают задачу доставки данных от одного узла сети (хоста), другому узлу, как правило, находящемуся в другой, удаленной сети. Эта функция в первом приближении соответствует транспортному уровню.

Интернет протокол (IP), и протокол контроля передачи (ICMP) — называются интернет-протоколами (просим прощения за тавтологию). Эти протоколы являются основой всего стека протоколов и отвечают за доставку данных между узлами в составных гетерогенных сетях. IP датаграммы являются конвертами для блоков данных TCP и UDP. В то время как TCP и

Прикладной

Представления



UDP заголовки обрабатываются только на конечных узлах, то IP заголовки обрабатываются всеми промежуточными узлами, через которые осуществляется доставка. Таким образом, протоколы TCP (вместе с UDP) имеют такое же отношение к IP, как груз к транспортному средству.

Расшифруем название основных протоколов и модулей, указанных на рисунке 4.1.

**IP (Internet Protocol) —** Интернет протокол. Базовый транспортный протокол стека. Передача данных осуществляется в конечном счете именно с помощью этого протокола в виде IP-датаграмм.

**ICMP (Internet Control Message Protocol)** — протокол контроля Интернет-сообщений. Выполняет служебную функцию. Оповещение о различных ситуациях, происходящих с IP-датаграммами.

**ARP (Address Resolution Protocol )** — протокол разрешения адресов. Позволяет определить IP-адрес по физическому (не IP) адресу узла. Необходим вследствие того, что у стека отсутствуют стандарты канального уровня.

**TCP (Transmission Control Protocol) —** протокол контроля передачи. Представляет собой протокол, ориентированный на соединения. Пользователь передает данный как бы по каналу, организованному TCP, но на самом деле TCP является надстройкой над IP.

**UDP (User Datagram Protocol)** — протокол пользовательских датаграмм. Предоставляет пользователю упрощенный относительно TCP сервис передачи данных.

**DNS (Domain Name System)** — система доменных имен. Функция аналогична по смыслу ARP, но производится на более высоком уровне. Здесь осуществляется получение IP-адреса хоста по его символьному имени.

**HTTP (Hypertext Transfer Protocol)** — протокол передачи гипертекста. Обеспечивает транспортировку гипертекстовых страниц в системе WWW.

**TELNET —** протокол эмуляции терминала. Позволяет осуществлять доступ к командной консоли удаленной машины.

FTP TELNET SMTP NFS

**SMTP (Simple Mail Transfer Protoco)** — простой протокол передачи почты. Обеспечивает отправку почты и доставку до почтового ящика пользователя.

**NFS (Network File System)** — сетевая файловая система. Набор протоколов, позволяющий Unix-машинам PC и ПК Macintosh, совместно использовать файлы в локальной сети.

Это лишь очень краткий перечень протоколов стека TCP/IP. Набор протоколов постоянно расширяется и модернизируется. Все изменения фиксируются в соответствующих документах RFC.

### 4.2.1 Адресация в TCP/IP и IPX/SPX

В любых сетях, не только в TCP/IP, необходимо идентифицировать отдельные, взаимодействующие между собой программы. Программы, в свою очередь, выполняются на конкретном компьютере, а компьютер размещается в некоторой локальной сети. Если предположить, что взаимодействующие программы выполняются на компьютерах, расположенных в разных локальных сетях, имеющих между собой межсетевое соединение, то получится, что полный сетевой адрес — это триада:

- адрес сети (подсети);
- адрес хоста (компьютера) в подсети;
- адрес программы на данном компьютере.

Любой стек протоколов должен так или иначе осуществлять адресацию трех вышеперечисленных объектов. Сравним реализацию этого принципа на примере двух наиболее популярных стеков TCP/IP и IPX/SPX.

В IPX/SPX адрес подсети выбирается администратором при установке сетевой интерфейсной карты в сервер. Размер номера подсети равен 4 байтам (8 шестнадцатеричных цифр). Пусть номером нашей гипотетической подсети будет шестнадцатиричное число: 0000ABBA. Главное, чтобы у разных подсетей были уникальные номера.

В качестве адреса хоста в IPX/SPX используется специальный адрес канального уровня — MAC-адрес. Этот адрес присваивается сетевой карте в момент производства. MAC-адреса уникальны для каждого сетевого устройства и используются



в большинстве современных сетевых технологий канального уровня: Ethernet, FDDI, ATM и др. Для всех существующих технологий ЛВС MAC-адрес имеет формат 6 байт, например,11:A0:17:3D:BC:01. По этому адресу можно однозначно идентифицировать компьютер.

Адрес программы или программного соединения, иногда называемый *сокет* (англ. socket — разъём), в IPX/SPX назначается автоматически, наподобие случайного числа. Пусть в нашем примере, это четыре шестнадцатиричные цифры или два байта: 405D.

Таким образом, некоторая сетевая программа, будет иметь полный сетевой адрес в виде: 0000ABBA:11A0173DBC01:405D.

Другая программа, запущенная на этом же компьютере, получит другой номер сокета, а две первые составляющие — адрес сети и адрес узла — останутся прежними. В IPX/SPX на одном компьютере может быть запущено 65536 различных сетевых программ с уникальными номерами сокета. На сегодняшний день этого более чем достаточно.

В TCP/IP адресация несколько сложнее. В стеке TCP/IP для адресации используются следующие объекты: IP-адреса, номера портов и символьные доменные имена. IP — это основной тип адресов, на основании которых сетевой уровень передает пакеты между сетями. IP версии 4 ($IP_v4$) занимает 4 байта, например, 109.26.17.100. Если IPX/IPX использует MAC-адреса канального уровня, то IP-адресация — это самостоятельная, независимая от технологий канального уровня, система адресации. Как мы уже говорили, это было сделано преднамеренно, так как TCP/IP предназначался изначально для объединения локальных сетей, использующих разнообразные технологии передачи данных, и, следовательно, нужна была самостоятельная система адресации, позволяющая уникально идентифицировать любой компьютер в глобальном масштабе. Несмотря на то, что в TCP/IP не рассматриваются технологии канального и физического уровней, при реальной передаче данных все равно приходится отображать IP адрес на адрес канального уровня. Например, отображение на MAC-адреса осуществляет Address Resolution Protocol (ARP).

IP назначается не производителем, а администратором сети во время конфигурирования компьютеров и маршрутизаторов. IP состоит из двух основных частей: номера сети и номера узла. Номер сети может быть выбран произвольно администратором, либо назначен по рекомендации специального подразделения InterNIC (Internet Network Information Center), если подсеть должна работать как составная часть Internet. Поставщики услуг Internet (или провайдеры) получают диапазоны IP-адресов, а затем распределяется между абонентами сети. IP — не зависит от локального адреса компьютера. Маршрутизатор по определению входит в несколько сетей, поэтому каждое соединение имеет свой IP. Конечный узел может также входить в несколько сетей, поэтому справедливо было бы сказать, что отдельный IP-адрес характеризует одно сетевое соединение.

Выше мы говорили о том, что полный сетевой адрес — это триада. В этом контексте IP адрес отвечает за идентификацию подсети и хоста в этой подсети. За идентификацию программы на данном компьютере отвечает номер порта. Номера портов — двухбайтные числа. Номера портов со значениями ниже 1024 называются популярными портами, они зарезервированы за стандартными сервисами.

Так, например, если клиент осуществил соединение с сервером на порту номер 25, то это значит, что пользователь желает отправить сообщение электронной почты. Тогда триада сетевого адреса в этом случае будет представлена в виде: 109.26.17.100:25. Сетевые запросы и ответы в глобальной сети всегда сопровождаются адресной информацией в виде: ‘IP-адрес:порт’.

Осталось выяснить, каким образом из 4-х байтного адреса выделяют адрес сети и адрес узла. Существует два основных подхода — это классы адресов и маски подсетей.

*Классы IP адресов*

IP — 4 байтное число. Состоит из двух частей — номера узла и номера сети. Какая часть адреса относится к узлу, а какая к сети зависит от значения первых бит адреса.

Различают 5 классов адресов.



Таблица 4.1

| Класс | № байта | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| A | 0.№сети. | №у | №у | №у |
| B | 10... №сети | | №у | №у |
| C | 110..... №сети | | | №у |
| D | 1110.... | Мультикаст адрес | | |
| E | 11110... | Зарезервирован | | |

Соответственно для классов адресов можно определить диапазон номеров сетей максимальное количество узлов в сети.

Таблица 4.2

| Класс | Наименьший номер сети | Наибольший номер сети | Максмальное число узлов в сети (степень 2) |
|---|---|---|---|
| A | 1.0.0.0 | 126.0.0.0 | 24 |
| B | 128.0.0.0 | 191.255.0.0 | 16 |
| C | 192.0.1.0 | 223.255.255.0 | 8 |
| D | 224.0.0.0 | 239.255.255.255 | Multicast |
| E | 240.0.0.0 | 247.255.255.255 | Зарезервирован |

*Особые IP-адреса*

1. Если весь адрес состоит только из нулей (0.0.0.0), то он обозначает адрес того узла, который сгенерировал этот пакет;

2. если в поле номера сети стоят одни нули, то по умолчанию считается, что узел назначения находится в той же сети, что и узел отправитель;

3. если все двоичные разряды IP адреса равны 1 (255.255.255.255), то пакет должен рассылаться всем узлам той же сети, что и отправитель. Такая рассылка называется ограниченным широковещательным сообщением (только внутри сети);

4. если в поле номера узла назначения стоят только единицы, то пакет рассылается всем узлам с данным номером сети (рассылка извне).

Особый смысл имеет IP-адрес, первый октет которого равен 127. Когда программа посылает данные по адресу 127.0.0.1, то образуется петля. Данные не передаются по сети, а возвращаются модулям стека верхнего уровня, как только что принятые.

Этот адрес называется loopback и служит для отладки и тестирования межпроцессных взаимодействий внутри одной машины.

Мультикаст адрес предназначен для образования произвольных групп адресов в сетях. Хост, который хочет передавать информацию группе адресов, сообщает о создании новой мультивещательной группы. Хосты, которые хотят присоединиться к новой группе, сообщают об этом своим маршрутизаторам, а те, в свою очередь, хосту-инициатору. Это может быть полезным для передачи по типу один ко многим, как, например, вещание видео и аудио программ.

На данный момент существуют только экспериментальные разработки, но в будущем это может составить конкуренцию радио и телевидению.

*Использование масок подсетей*

Рассмотрим адрес 185.23.44.206. Этот IP попадает в диапазон 128-191, и мы можем сказать, что этот адрес относится к классу B, а значит номером сети являются первые два байта 185.23.0.0, а номером узла 0.0.44.206.

Часто на практике используются маски, добавляющие гибкости в схему IP адресации. Маска — это число, которое используется в паре с IP адресом, двоичная запись маски содержит единицы в тех разрядах, которые должны в IP-адресе интерпретироваться как номер сети. Для стандартных классов маски имеют следующие значения.

A 11111111.00000000.00000000.00000000 (255.0.0.0)
B 11111111.11111111.00000000.00000000 (255.255.0.0)
C 11111111.11111111.11111111.00000000 (255.255.255.0)

Снабжая каждый адрес маской, можно отказаться от понятия классов, и расширить количество адресуемых сетей.

Если рассматривать адрес 185.23.44.206 совместно с маской 255.255.255.0, то номером сети будет 185.23.44.0, а не 185.23.0.0.

А если использовать маску 255.255.240.0, то номером сети будет 185.23.32.0, а номером узла 0.0.12.206.

185.23.44.206 соответствует 10111001.00010111.00101100.11001110
255.255.240.0 соответствует 11111111.11111111.11110000.00000000



Чтобы выделить адрес подсети по маске, необходимо двоичное представление IP-адреса побитно умножить на двоичное представление маски. Адрес хоста можно получить путем вычитания полученного адреса подсети из начального IP-адреса.

Механизм масок широко распространен в IP-маршрутизации. Администраторы могут по-своему структурировать сеть, не используя дополнительные классы адресов. Если у вас есть блок из 500 адресов, маловероятно, что они все будут использоваться в одном сетевом сегменте, скорее всего, адреса будут распределены по разным отделам или компьютерным классам. И тогда администратор, используя механизм назначения масок, может сам структурировать сеть по своему усмотрению.

Уже сравнительно давно наблюдается дефицит адресов. Очень трудно получить адреса класса B и практически невозможно стать обладателем адреса класса A. Дефицит обусловлен не только ростом сетей, но и нерациональным использованием имеющихся адресов, так часто владельцы адресов класса C используют лишь часть из 254. Также возникает проблема вырожденных сетей при соединении двух маршрутизаторов соединением типа «точка-точка». При этом для вырожденной сети, в которой соединяются только смежные порты двух маршрутизаторов, приходится выделять отдельный номер сети, хотя она содержит только два узла.

Если же IP-сеть работает в автономном режиме, без подключения к глобальной сети, то есть без соединения в Internet, тогда администратор может выбрать номера произвольно. Но и в этой ситуации, для того чтобы избежать коллизий, в стандартах Internet определено несколько диапазонов адресов, рекомендуемых для локального, частного, использования. Эти адреса не обрабатываются маршрутизаторами Internet ни при каких условиях, даже если сеть подключена к глобальной сети. В классе A это сеть 10.0.0.0, в классе B — это диапазон 172.16.0.0-172.31.0.0, в классе C — 192.168.0.0-192.168.255.0

Другая технология, которая используется для снятия проблемы нехватки адресов, — это трансляция адресов — Network Address Translation (NAT). Узлам сети адреса назначаются произвольно, словно сеть работает автономно. Внутренняя сеть соединяется с Интернет через некоторое промежуточное устрой-

ство (шлюз, маршрутизатор, межсетевой экран). Данное устройство обладает некоторым количеством внешних «легальных» IP-адресов, согласованных с поставщиком услуг Интернет. Промежуточное устройство способно преобразовывать внутренние адреса во внешние, используя для этого таблицы соответствия. Процедура трансляции адресов определена в RFC 1631.

*Автоматизация процесса назначения IP-адресов*

Назначение IP-адресов узлам сети даже при не очень большом размере сети может быть достаточно утомительной процедурой. Протокол DHCP (Dynamic Host Configuration Protocol) автоматизирует процесс назначения адресов. DHCP работает в клиент-серверном режиме. Во время старта системы компьютер, являющийся DHCP-клиентом, посылает в сеть широковещательный запрос на получение IP-адреса. DHCP-сервер отвечает на запрос сообщением, содержащим IP-адрес. При динамическом распределении адресов IP-адрес выделяется на ограниченное время. В этом случае количество узлов сети может превышать количество адресов. В ручной процедуре администратор создает таблицу соответствия адресов. А DHCP распределяет их клиентам в статическом режиме одному клиенту один и тот же адрес. При автоматическом распределении DHCP присваивает адрес из набора определенных адресов без вмешательства оператора. Администратор только определяет диапазон допустимых адресов. Кроме IP, можно назначить и другие параметры стека, например, маску, адрес маршрутизатора, адрес сервера DNS, доменное имя компьютера.

### 4.2.2 Система доменных имен DNS

Все уровни, находящиеся в модели OSI ниже прикладного, служат для обеспечения надежной доставки данных, но никаких полезных для пользователя действий не производят. Изучим некоторые реальные сетевые приложения.

Разумеется, даже прикладной уровень нуждается в обслуживающих протоколах, с помощью которых осуществляется функционирование приложений. Соответственно, прежде чем начать рассмотрение самих приложений, мы изучим один из таких протоколов. Речь идет о службе имен доменов DNS, обеспечивающей присвоение имен в Интернете. Затем мы рассмотрим два реально действующих приложения: электронную почту и Всемирную паутину.



*Служба имен доменов DNS*

Хотя программы теоретически могут обращаться к хостам, почтовым ящикам и другим ресурсам по их сетевым адресам (например, IP), пользователям запоминать их тяжело. Кроме того, отправка электронной почты на адрес Tanya@128.111.24.41 будет означать, что в случае переезда сервера таниного провайдера или организации на новое место с новым IP-адресом придется изменить ее адрес e-mail. Для отделения имен машин от их адресов было решено использовать текстовые ASCII-имена. Поэтому танин адрес более привычно выглядит в таком виде: Tanya@art.ucsb.edu. Тем не менее, сеть сама по себе понимает только численные адреса, поэтому нужен механизм преобразования ASCII-строк в сетевые адреса.

Когда-то давно в сети ARPANET соответствие между текстовыми и двоичными адресами просто записывалось в файле *hosts.txt,* в котором перечислялись все хосты и их IP-адреса. Каждую ночь все хосты получали этот файл с сайта, на котором он хранился. В сети, состоящей из нескольких сотен больших машин, работающих под управлением системы с разделением времени, такой подход работал вполне приемлемо.

Но когда к сети подключились тысячи рабочих станций, всем стало ясно, что этот способ не сможет работать вечно. Во-первых, размер файла рано или поздно стал бы слишком большим. Однако, что еще важнее, если управление именами хостов не осуществлять централизованно, неизбежно возникновение конфликтов имен. В то же время, представить себе централизованное управление именами всех хостов гигантской международной сети довольно сложно. Для разрешения всех этих проблем и была разработана служба имен доменов (DNS, Domain Name System).

Суть системы DNS заключается в иерархической схеме имен, основанной на доменах, и распределенной базе данных,

реализующей эту схему имен. В первую очередь эта система используется для преобразования имен хостов и пунктов назначения электронной почты в IP-адреса, но также может использоваться и в других целях. Определение системы DNS дано в RFC 1034 и 1035.

В общих чертах система DNS применяется следующим образом. Для преобразования имени в IP-адрес прикладная программа обращается к библиотечной процедуре, называющейся распознавателем, передавая ей имя в качестве параметра. Распознаватель посылает UDP-пакет локальному DNS-серверу, который ищет имя в базе данных и возвращает соответствующий IP-адрес распознавателю, который, в свою очередь, передает этот адрес вызвавшей его прикладной программе. Имея IP-адрес, программа может установить TCP-соединение с адресатом или послать ему UDP-пакеты.

*Пространство имен DNS*

Управление большим и постоянно изменяющимся набором имен представляет собой нетривиальную задачу. В почтовой системе на письмах требуется указывать (явно или неявно) страну, штат или область, город, улицу, номер дома, квартиру и фамилию получателя. Благодаря использованию такой иерархической схемы, не возникает путаницы между Марвином Андерсоном, живущим на Мейн-стрит в Уайт Плейнс, штат Нью-Йорк, и Марвином Андерсоном с Мейн-стрит в Остине, штат Техас. Система DNS работает аналогично.

Интернет концептуально разделен на 200 доменов верхнего уровня. Доменами называют в Интернете множество хостов, объединенных в логическую группу. Каждый домен верхнего уровня подразделяется на поддомены, которые, в свою очередь, также могут состоять из других доменов, и т.д. Все эти домены можно рассматривать в виде дерева (рис. 4.2). Листьями дерева являются домены, не разделяющиеся на поддомены (но состоящие из хостов, конечно). Такой конечный домен может состоять из одного хоста или может представлять компанию и содержать в себе тысячи хостов.



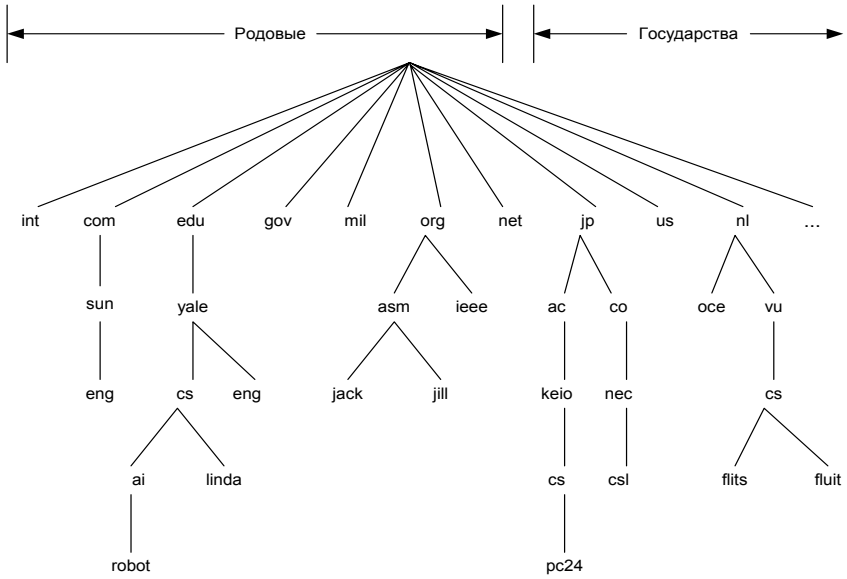



Рисунок 4.2 — Часть доменного пространства имен Интернета

Домены верхнего уровня разделяются на две группы: родовые домены и домены государств. К родовым относятся домены *com* (commercial — коммерческие организации), *edu* (educational — учебные заведения), *gov* (government — федеральное правительство США), *int* (international — определенные международные организации), *mil* (military — вооруженные силы США), *net* (network — сетевые операторы связи) и *org* (некоммерческие организации). За каждым государством в соответствии с международным стандартом ISO 3166 закреплен один домен государства.

В ноябре 2000 года ICANN было утверждено 4 новых родовых имени доменов верхнего уровня, а именно: *biz* (бизнес), *info* (информация), *name* (имена людей) и *pro* (специалисты, такие как доктора и адвокаты). Кроме того, по просьбе соответствующих отраслевых организаций были введены еще три специализированных имени доменов верхнего уровня: *aero* (аэрокосмическая промышленность), *coop* (кооперативы) и *museum* (музеи). В будущем появятся и другие домены верхнего уровня.

В принципе, получить домен второго уровня типа *name-of-company.com* несложно. Надо лишь проверить, не занято ли желаемое имя домена кем-то другим и не является ли оно чьей-нибудь торговой маркой. Для этого надо зайти на сайт регистрационного бюро верхнего уровня (в данном случае *com)*. Если все в порядке, заказчик регистрируется и за небольшую ежегодную абонентскую плату получает домен второго уровня. На сегодняшний день в качестве имен поддоменов *com* уже используются практически все общеупотребительные английские слова. Попробуйте набрать какое-нибудь слово, касающееся домашнего хозяйства, животных, растений, частей тела и т.д. Вряд ли ошибетесь.

Имя каждого домена, подобно полному пути к файлу в файловой системе, состоит из пути от этого домена до (безымянной) вершины дерева. Компоненты пути разделяются точками. Так, домен технического отдела корпорации Sun Microsystems может выглядеть как *eng.sun.com,* а не так, как это принято в стиле UNIX *(/com/sun/eng)*. Следует отметить, что *eng.sun.com* не конфликтует с потенциальным использованием имени *eng* в домене *eng.yale.edu,* где он может обозначать факультет английского языка Йельского университета.

Имена доменов могут быть абсолютными и относительными. Абсолютное имя домена всегда оканчивается точкой (например, *eng.sun.com.),* тогда как относительное имя — нет. Для того чтобы можно было единственным образом определить истинные значения относительных имен, они должны интерпретироваться в некотором контексте. В любом случае именованный домен означает определенный узел дерева и все узлы под ним.

Имена доменов нечувствительны к изменению регистра символов. Так, например, *edu* и *EDU* означают одно и то же. Длина имен компонентов может достигать 63 символов, а длина полного пути не должна превосходить 255 символов.

В принципе, новые домены могут добавляться в дерево двумя разными путями. Например, *cs.yale.edu* можно без проблем поместить в домен *us* под именем *cs.yale.ct.us*. На практике, однако, почти все организации в США помещаются под родовыми доменами, тогда как почти все организации за пределами Соединенных Штатов располагаются под доменами их государств. Не существует каких-либо правил, запрещающих регистрацию под двумя доменами верхнего уровня, однако использует эту возможность лишь небольшое число организаций (за исключением интернациональных, например, *sony.com* и *sony.nl.*



Каждый домен управляет доступом к доменам, расположенным под ним. Например, в Японии домены *ac.jp* и *co.jp* соответствуют американским доменам *edu* и *com.* В Голландии подобное различие не используется, и все домены организаций помещаются прямо под доменом *nl.* В качестве примера приведем имена доменов факультетов компьютерных наук трех университетов.

1. cs.yale.edu (Йельский университет, США).
2. cs.vu.nl (университет Врийе, Нидерланды).
3. cs.keio.ac.jp (университет Кейо, Япония).

Для создания нового домена требуется разрешение домена, в который он будет включен. Например, если в Йеле образовалась группа VLSI, которая хочет зарегистрировать домен *vlsi.cs.yale.edu,* ей нужно разрешение от того, кто управляет доменом *cs.yale.edu.* Аналогично, если создается новый университет, например, университет Северной Южной Дакоты, он должен попросить менеджера домена *edu* присвоить их домену имя *unsd.edu.* Таким образом, удается избежать конфликта имен, а каждый домен отслеживает состояние всех своих поддоменов. После того как домен создан и зарегистрирован, в нем могут создаваться поддомены, например, *cs.unsd.edu,* для чего уже не требуется разрешения вышестоящих доменов.

Структура доменов отражает не физическое строение сети, а логическое разделение между организациями и их внутренними подразделениями. Так, если факультеты компьютерных наук и электротехники располагаются в одном здании и пользуются одной общей локальной сетью, они, тем не менее, могут иметь различные домены. И наоборот, если, скажем, факультет компьютерных наук располагается в двух различных корпусах университета с различными локальными сетями, логически все хосты обоих зданий обычно принадлежат к одному и тому же домену.

*Записи ресурсов*

У каждого домена, независимо от того, является ли он одиноким хостом или доменом верхнего уровня, может быть набор ассоциированных с ним записей ресурсов. Для одинокого хоста запись ресурсов чаще всего представляет собой просто его IP-адрес, но существует также много других записей ресурсов. Когда распознаватель передает имя домена DNS-серверу, то, что он получает обратно, представляет собой записи ресурсов, ассоциированные с его именем. Таким образом, истинное назначение системы DNS заключается в преобразовании доменных имен в записи ресурсов.

Запись ресурса состоит из пяти частей. Хотя для эффективности они часто перекодируются в двоичную форму, в большинстве описаний записи представляются в виде ASCII-текста, по одной строке на запись ресурса. Мы будем использовать следующий формат:

Domain_name Time_to_live Class Type Value

Поле *Domain_name* (имя домена) обозначает домен, к которому относится текущая запись. Обычно для каждого домена существует несколько записей ресурсов, и каждая копия базы данных хранит информацию о нескольких доменах. Поле имени домена является первичным ключом поиска, используемым для выполнения запросов. Порядок записей в базе данных значения не имеет. В ответ на запрос о домене возвращаются все удовлетворяющие запросу записи требуемого класса.

Поле *Time_to_live* (время жизни) указывает, насколько стабильно состояние записи. Редко меняющимся данным присваивается высокое значение этого поля, например, 86 400 (число секунд в сутках). Непостоянная информация помечается небольшим значением, например, 60 (1 минута).

Третьим полем каждой записи является поле *Class* (класс). Для информации Интернета значение этого поля всегда равно *IN.* Для прочей информации применяются другие коды, однако на практике они встречаются редко.

Запись *SOA* (Start Of Authority — начальная точка полномочий) сообщает имя первичного источника информации о зоне сервера имен (описанного ниже), адрес электронной почты его



администратора, уникальный порядковый номер, различные флаги и тайм-ауты.

Самой важной является запись *A* (Address — адрес). Она содержит 32-разрядный IP-адрес хоста. У каждого хоста в Интернете должен быть по меньшей мере один IP-адрес, чтобы другие машины могли с ним общаться. На некоторых хостах может быть одновременно установлено несколько сетевых соединений. В этом случае им требуется по одной записи типа *A* для каждого сетевого соединения (для каждого IP-адреса). DNS можно настроить на циклический перебор этих записей, чтобы в ответ на первый запрос возвращалась первая запись, в ответ на второй запрос — вторая запись и т.д.

Следующей по важности является запись *MX.* В ней указывается имя хоста, готового принимать почту для указанного домена. Дело в том, что не каждая машина может заниматься приемом почты. Если кто-нибудь хочет послать письмо на адрес, например, bill@microsoft.com, то отправляющему хосту нужно будет вначале найти почтовый сервер на microsoft.com. Запись *MX* может помочь в этих поисках.

Записи *NS* содержат информацию о серверах имен. Например, в каждой базе данных DNS содержится *NS-запись* для каждого домена верхнего уровня, что позволяет пересылать электронную почту на удаленные участки дерева имен.

Записи *CNAME* позволяют создавать псевдонимы. Представим себе, что человек, знакомый в общих чертах с формированием имен в Интернете, хочет послать сообщение человеку с регистрационным именем *paul* на отделении компьютерных наук Массачусетского технологического института (M.I.T.). Он может попытаться угадать нужный ему адрес, составив строку *paul@cs.mit.edu.* Однако этот адрес работать не будет, так как домен отделения компьютерных наук Массачусетского технологического института на самом деле называется *lcs.mit.edu.* Таким образом, для удобства тех, кто этого не знает, M.I.T. может создать запись *CNAME,* позволяющую обращаться к нужному домену по обоим именам. Такая запись будет иметь следующий вид:

cs.mit.edu 86400 IN CNAME lcs.mit.edu

Как и *CNAME,* запись PTR указывает на другое имя. Однако в отличие от записи *CNAME,* являющейся, по сути, макроопределением, PTR представляет собой регулярный тип данных DNS, интерпретация которого зависит от контекста. На практике запись PTR почти всегда используется для ассоциации имени с IP-адресом, что позволяет по IP-адресу находить имя соответствующей машины. Это называется обратным поиском.

Запись *HINFO* позволяет определять тип машины и операционной системы, которой соответствует домен. Наконец, TXT-записи позволяют доменам идентифицировать себя произвольным образом. Оба эти типа записей разработаны для удобства пользователей. Ни один из них не является обязательным, поэтому рассчитывать на их наличие не следует, особенно при обработке записей программами (тем более что программы практически невозможно научить обрабатывать эти текстовые данные).

Наконец, последнее поле записи ресурса — это поле *Value* (значение). Это поле может быть числом, именем домена или текстовой ASCII-строкой. Смысл поля зависит от типа записи. Краткое описание поля *Value* для каждого из основных типов записей дано в таблице 4.3.

Поле *Type* (тип) означает тип записи.

Таблица 4.3

| Тип (*Type*) | Смысл | Значение (*Value*) |
|---|---|---|
| SOA | Начальная запись зоны | Параметры для этой зоны |
| A | IP-адрес хоста | 32-разрядное целое число |
| MX | Обмен почтой | Приоритет, с которым домен желает принимать электронную почту |
| NS | Сервер имен | Имя сервера для этого домена |
| CNAME | Каноническое имя | Имя домена |
| PTR | Указатель | Псевдоним IP-адреса |
| HINFO | Описание хоста | Описание центрального процессора и ОС в виде ASCII-текста |
| TXT | Текст | Не интерпретируемый ASCII-текст |



Пример информации, хранящейся в базе данных DNS домена, приведен в листинге 4.1. В нем показана часть (почти что гипотетической) базы данных домена *cs.vu.nl,* представленного также в виде узла дерева доменов на рис. 4.2. В базе данных содержится семь типов записей ресурсов.

Листинг 4.1 Часть возможной базы данных домена cs.vu.nl

```
: Официальная информация для cs.vu.nl
cs.vu.nl 86400 IN SOA star boss (952771.7200.7200.2419200.86400)
cs.vu.nl 86400 IN TXT “Faculteit Wiskunde en Informatica.”
cs.vu.nl 86400 IN TXT “Vrije Universiteit Amsterdam.”
cs.vu.nl 86400 IN MX 1 zephyr.cs.vu.nl.
cs.vu.nl 86400 IN MX 2 top.cs.vu.nl.
flits. cs.vu.nl. 86400 IN HINFO Sun Unix
flits. cs.vu.nl. 86400 IN A 130.37.16.112
flits. cs.vu.nl. 86400 IN A 192.31.231.165
flits. cs.vu.nl. 86400 IN MX 1 flits.cs.vu.nl.
flits. cs.vu.nl. 86400 IN MX 2 zephyr.cs.vu.nl.
flits. cs.vu.nl. 86400 IN MX 3 top.cs.vu.nl.
www.cs.vu.nl. 86400 IN CNAME star.cs.vu.nl.
ftp.cs.vu.nl. 86400 IN CNAME zephyr.cs.vu.nl.
rowboat IN A 130.37.56.201
IN MX 1 rowboat
IN MX 2 zephyr
IN HINFO Sun Unix
Little-sister IN A 130.37.62.23
IN HINFO Mac MacOS
Laserjet IN A 192.31.231.216
IN HINFO “HP Laserjet IIISi” Proprietary
```

В первой строке листинга, не являющейся комментарием, дается основная информация о домене, которая в дальнейшем нас интересовать не будет. В следующих двух строках приводится текстовая информация об организации, которой принадлежит домен. Следующие две строки определяют два хоста, с которыми следует связаться в первую очередь при попытке доставить электронную почту, посланную по адресу person@cs.vu.nl. Хост по имени *zephyr* (специальная машина) следует опросить первым. В случае неудачи следует попробовать доставить письмо машине по имени *top.*

После пустой строки, добавленной для удобства чтения, следуют строки, сообщающие о том, что хост *flits* является рабочей станцией Sun, работающей под управлением операционной системы UNIX, а также даются оба ее IP-адреса. Следующие три строки указывают хосты, которым следует доставлять письма, посылаемые по адресу *flits.cs.vu.nl.* В первую очередь, естественно, следует пытаться доставить письмо самому компьютеру *flits.* Но если этот хост выключен, следует продолжать попытки, обращаясь к хостам *zephyr* и *top.* Следом указаны псевдонимы *www.cs.vu.nl* и *ftp.cs.vu.nl,* позволяющие домену *cs.vu.nl* изменять свой WWW и FTP-серверы, не меняя адресов, по которым пользователи смогут продолжать к ним обращаться.

Следующие четыре строки содержат обычные записи для рабочих станций, в данном случае для *rowboat.cs.vu.nl.* Хранящаяся в базе данных информация содержит IP-адрес, имена первого и второго хостов для доставки почты и информацию о машине. Следом идут две записи о машине *little-sister,* работающей под управлением системы MacOS, отличной от системы UNIX (поэтому эта машина не может сама получать электронную почту). Последние две строки описывают лазерный принтер, подключенный к Интернету.

В этом файле нет IP-адресов доменов верхнего уровня, так как они не принадлежат домену *cs.vu.nl.* Эти адреса поставляются корневыми серверами, чьи IP-адреса присутствуют в файле конфигурации системы и загружаются в DNS-кэш, когда загружается DNS-сервер. Существует около дюжины корневых серверов по всему миру, и каждый из них знает IP-адреса всех серверов доменов верхнего уровня. То есть если машине известен IP-адрес хотя бы одного корневого сервера, она может узнать любое имя DNS.

*Серверы имен*

Теоретически один сервер мог бы содержать всю базу данных DNS и отвечать на все запросы к ней. На практике этот сервер оказался бы настолько перегруженным, что был бы просто бесполезным. Более того, если бы с ним когда-нибудь что-нибудь случилось, то весь Интернет не работал бы.



Чтобы избежать проблем, связанных с хранением всей информации в одном месте, пространство имен DNS разделено на непересекающиеся зоны (рис. 4.3). Каждая зона содержит часть общего дерева доменов, а также в нее входят серверы имен, хранящие управляющую информацию об этой зоне. Обычно в каждой зоне находится один основной сервер зоны, получающий информацию из файла на своем диске, и несколько дополнительных серверов имен, которые получают информацию от основного сервера имен. Для большей надежности некоторые серверы, обслуживающие зону, могут находиться за пределами самой зоны.

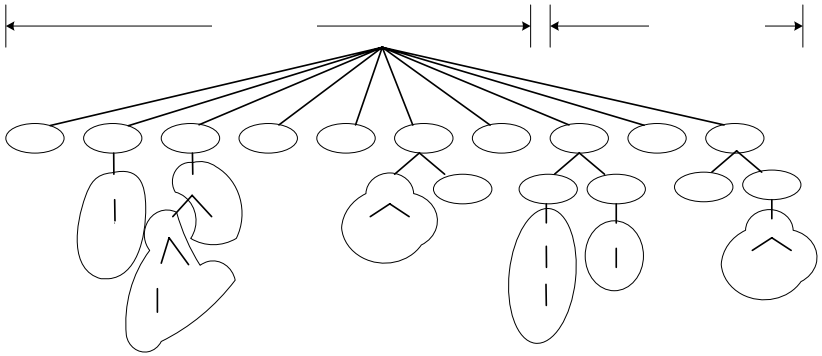


Рисунок 4.3 — Часть пространства имен DNS, разделенная на зоны

Расстановка границ зон целиком зависит от администратора зоны. Это решение основывается на том, сколько серверов имен требуется в той или иной зоне. Например, у Йельского университета (рис. 4.3) есть сервер для *yale.edu,* управляющий доменом *eng.yale.edu,* но не доменом *cs.yale.edu,* расположенным в отдельной зоне со своими серверами имен. Подобное решение может быть принято, когда факультет английского языка не хочет управлять собственным сервером имен, но этого хочет факультет компьютерных наук. Соответственно, домен *cs.yale.edu* выделен в отдельную зону, а домен *eng.yale.edu* — нет.

Распознаватель обращается с запросом разрешения имени домена к одному из локальных серверов имен. Если искомый

домен относится к сфере ответственности данного сервера имен, как, например, домен *ai.cs.yale.edu* подпадает под юрисдикцию домена *cs.yale.edu,* тогда данный DNS-сервер сам отвечает распознавателю на его запрос, передавая ему авторитетную запись ресурса. Авторитетной называют запись, получаемую от официального источника, хранящего данную запись и управляющего ее состоянием. Поэтому такая запись всегда считается верной, в отличие от кэшируемых записей, которые могут устаревать.

Однако если домен удаленный, и информацию о запрашиваемом домене нельзя получить от данного сервера имен, последний посылает сообщение с запросом серверу домена верхнего уровня запрашиваемого домена. Рассмотрим данный процесс на примере (рис. 4.4). В данном случае распознаватель на компьютере *flits.cs.vu.nl* хочет узнать IP-адрес хоста *linda.cs.yale.edu.* На первом шаге он посылает запрос локальному серверу имен, *cs.vu.nl.* Этот запрос содержит имя искомого домена, тип *(A)* и класс *(IN).*

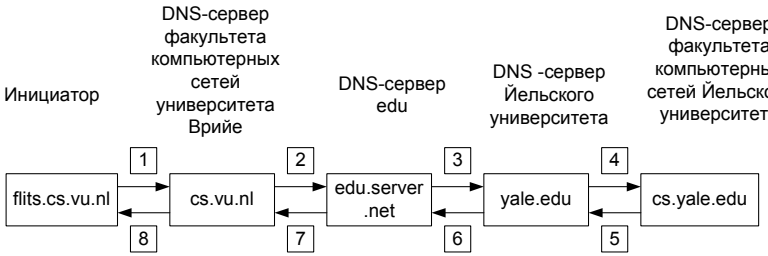



Рисунок 4.4 — Восемь этапов поиска распознавателем имени удаленного хоста

Предположим, что локальный сервер имен никогда ранее не получал запроса об этом домене и поэтому ничего о нем не знает. Он может опросить несколько находящихся рядом серверов имен, но если они также не знают ответа, данный локальный сервер посылает UDP-пакет серверу домена *edu,* адрес которого содержится в его базе данных. Как видно из рисунка, его имя *edu-seroer.net.* Маловероятно, чтобы этот сервер знал адрес хоста *linda.cs.yale.edu.* Скорее всего он даже не знает адреса сервера



*cs.yale.edu,* однако он должен знать все свои дочерние домены, поэтому он направляет запрос серверу имен домена *yale.edu* (шаг 3). Тот, в свою очередь, пересылает этот запрос серверу *cs.yale.edu* (шаг 4), у которого должны быть требуемые авторитетные записи ресурсов. Поскольку каждый запрос был от клиента к серверу, посылаемые в ответ записи ресурсов проделывают тот же путь в обратном направлении (шаги 5-8).

Когда записи ресурсов попадают на сервер имен *cs.vu.nl,* они помещаются в кэш на случай, если они понадобятся еще раз. Однако эта информация не является авторитетной, так как изменения в домене *cs.yale.edu* не будут распространяться автоматически на все кэши, в которых может храниться копия этой информации. По этой причине записи кэша обычно долго не живут. В каждой записи ресурса присутствует поле *Time_to_live.* Оно сообщает удаленным серверам, насколько долго следует хранить эту запись в кэше. Если какая-либо машина сохраняет постоянный адрес годами, возможно, будет достаточно надежно хранить эту информацию в кэше в течение одного дня. Для более непостоянной информации, вероятно, более осмотрительно удалять все записи через несколько секунд или одну минуту. Практика показывает, что удобнее, наоборот, хранить эту информацию на своем хосте, для чего существуют специальные программы, поддерживающие мини-базы DNS для часто используемых адресов на компьютерах пользователя.

Следует отметить, что серия запросов, описанная выше, называется рекурсивным запросом, так как каждый сервер, который не владеет запрашиваемой информацией, передает запрос дальше, а затем ответ, также поэтапно, пересылается обратно. Возможны и другие схемы реализации запросов. Так, например, возможна форма запроса, в которой в случае отсутствия требуемой информации на локальном сервере клиент сразу получает сообщение о неуспешном выполнении запроса, но при этом ему сообщается имя следующего сервера, который можно об этом спросить. Такая процедура, дающая клиенту больше контроля над процессом поиска, реализована на некоторых серверах. Некоторые серверы вообще не выполняют рекурсивных запросов, а всегда возвращают имя сервера для следующего запроса.

Также следует сказать, что когда DNS-клиент не получает ответа на посланный запрос, он обычно пытается получить искомую информацию у другого сервера, прежде чем истечет период ожидания. Это делается, исходя из предположения о том, что сервер, которому адресовался запрос, может в данный момент оказаться выключенным (а не о том, что запрос или ответ потерялся).

Хотя система DNS чрезвычайно важна для обеспечения корректной работы Интернета, основная ее задача заключается в отображении текстовых имен машин на пространство их IP-адресов. Она не помогает найти людей, ресурсы, услуги или другие объекты. Для этого существует иная услуга, называемая LDAP (Light-weight Directory Access Protocol — облегченный протокол службы каталогов). Это упрощенная версия стандартной службы каталогов OSI X.500, описание LDAP содержится в RFC 2251. Информация организуется в древовидной форме, и по этому дереву можно осуществлять поиск различных компонентов. Это напоминает телефонную книгу наподобие «желтые страницы».

### 4.2.3 Интернет протокол-IP

Internet protocol (IP) описан в документе RFC791. Рассмотрим более подробно, что представляет собой Интернет протокол.

Как уже было сказано в начале раздела 4.2, в сети непосредственно передаются IP-датаграммы, состоящие из заголовка и данных. В поле данных размещается служебная информация вышестоящих протоколов (TCP и UDP) и данные пользователя.

IP специально ограничен задачами обеспечения функций, необходимых для передачи битового пакета (датаграммы Internet) от отправителя к получателю через объединенную систему компьютерных сетей. Нет механизмов проверки достоверности конечных данных, управления протоколом, синхронизации или других услуг, обычно применяемых в протоколах передачи от хоста к хосту. Протокол Internet может обобщить услуги поддерживающих его сетей с целью предоставления услуг различных типов и качеств.



IP выполняет две главные функции: адресацию и фрагментацию. Программные модули стека TCP/IP (далее просто стек TCP/IP) используют адреса, помещенные в заголовок IP пакета, для передачи Internet датаграмм их получателям. Выбор пути передачи называется маршрутизацией.

Стек TCP/IP используют поля в заголовке датаграммы для фрагментации и восстановления датаграмм, когда это необходимо для их передачи через сети с малым размером пакетов.

Сценарий действия состоит в том, что стек TCP/IP меняет размер датаграммы на каждом из хостов, задействованных в internet-коммуникации и на каждом из шлюзов, обеспечивающих взаимодействие между сетями. Стек на каждом из хостов придерживается общих правил для интерпретации полей адресов, для фрагментации и сборки Internet датаграмм. Кроме этого, хосты (и особенно шлюзы) имеют процедуры для принятия решений о маршрутизации, а также другие функции.

Протокол Internet обрабатывает каждую датаграмму как независимую единицу, не имеющую связи ни с какими другими датаграммами Internet. Протокол не имеет средств установления соединения.

Протокол Internet использует четыре ключевых механизма для формирования своих услуг: задание типа сервиса, времени жизни, опций и контрольной суммы заголовка.

Тип обслуживания используется для обозначения требуемой услуги. Тип обслуживания - это абстрактный или обобщенный набор параметров, который характеризует набор услуг, предоставляемых сетями, и составляющих собственно протокол Internet. Этот способ обозначения услуг должен использоваться шлюзами для выбора рабочих параметров передачи в конкретной сети, для выбора сети, используемой при следующем переходе датаграммы, для выбора следующего шлюза при маршрутизации сетевой датаграммы.

Механизм времени жизни служит для указания верхнего предела времени жизни Internet датаграммы. Этот параметр устанавливается отправителем датаграммы и уменьшается в каждой точке на проходимом датаграммой маршруте. Если параметр времени жизни станет нулевым до того, как Internet датаграмма достигнет получателя, эта датаграмма будет уничтоже-

на. Время жизни можно рассматривать как часовой механизм самоуничтожения.

Механизм опций предоставляет функции управления, которые являются необходимыми или просто полезными при определенных ситуациях, однако он не нужен при обычных коммуникациях. Механизм опций предоставляет такие возможности, как временные штампы, безопасность, специальная маршрутизация.

Контрольная сумма заголовка обеспечивает проверку того, что информация, используемая для обработки датаграмм Internet, передана правильно. Данные могут содержать ошибки. Если контрольная сумма неверна, то Internet датаграмма будет разрушена, как только ошибка будет обнаружена.

Схему действий для передачи датаграммы от одной прикладной программы к другой можно проиллюстрировать следующим образом (рис. 4.5 ):

Рисунок 4.5 — Путь передачи датаграммы

Предположим, что перенос будет включать прохождение одного промежуточного шлюза. Отправляющая прикладная программа на хосте А готовит свои данные и вызывает свой локальный стек Internet для отправки этих данных в качестве датаграммы, а в качестве аргументов этого вызова передает адрес получателя и другие параметры.



Стек TCP/IP готовит заголовок датаграммы и стыкует с ним данные, затем определяет локальный сетевой адрес, соответствующий данному адресу Internet. В данном случае это адрес шлюза.

Модуль передает данную датаграмму и адрес в локальной сети в распоряжение интерфейса локальной сети.

Интерфейс локальной сети создает соответствующий этой сети заголовок и соединяет с ним датаграмму. Затем он передает по локальной сети полученный таким образом результат.

Датаграмма достигает хост-компьютера, играющего роль шлюза и расположенного в вершине сети. Интерфейс локальной сети отделяет этот заголовок и передает датаграмму на стек TCP/IP. Стек TCP/IP определяет из Internet адреса, что датаграмма должна быть направлена на хост-компьютер во второй сети. Стек TCP/IP определяет адрес хоста-получателя в локальной сети. Он обращается к интерфейсу локальной сети с тем, чтобы она переслала данную датаграмму по назначению.

Интерфейс создает заголовок локальной сети и соединяет с ним датаграмму, а затем результат направляет на хост-получатель.

На хосте-получателе интерфейс локальной сети удаляет заголовок локальной сети и передает оставшуюся часть на Internet модуль.

Стек TCP/IP определяет, что рассматриваемая выше датаграмма предназначена для прикладной программы на этом хосте. Модуль передает данные прикладной программе в ответ на системный вызов. В качестве результата этого вызова передаются адрес получателя и другие параметры.

Функция или цель протокола IP состоит в передаче датаграммы через набор объединенных компьютерных сетей. Это осуществляется посредством передачи датаграмм от одного стека TCP/IP к другому до тех пор, пока не будет достигнут получатель. Экземпляры стека TCP/IP находятся на хостах и шлюзах системы Internet. Датаграммы направляются через конкретные компьютерные сети, основанные на интерпретации Internet адресов. Таким образом, одним из важных механизмов протокола Internet является Internet адрес.

При передаче сообщений с одного Internet модуля на другой датаграммы могут нуждаться в прохождении через сети, для которых максимальный размер пакета меньше, чем размер датаграммы. Чтобы преодолеть эту сложность, в протокол Internet включен механизм фрагментации.

Фрагментация Internet датаграммы необходима, когда эта датаграмма возникает в локальной сети, позволяющей работать с пакетами большого размера, и затем должна пройти к получателю через другую локальную сеть, которая ограничивает пакеты меньшим размером.

IP датаграмма может быть помечена как нефрагментируемая. Любая Internet датаграмма, помеченная таким образом, не может быть фрагментирована модулем Internet ни при каких условиях. Если же Internet датаграмма, помеченная как нефрагментируемая, тем не менее не может достигнуть получателя без фрагментации, то вместо этого она будет разрушена.

Фрагментация, перенос и сборка в локальной сети, невидимые для отправителя и получателя, называются внутрисетевой фрагментацией. Необходимо, чтобы Internet процедуры фрагментации и сборки могли разбивать датаграмму на почти любое количество частей, которые впоследствии могли бы быть вновь собраны. Получатель фрагмента использует поле идентификации для того, чтобы быть убежденным в том, что фрагменты различных датаграмм не будут перепутаны. Поле смещения фрагмента сообщает получателю положение фрагмента в исходной датаграмме.

Смещение фрагмента и длина определяют кусок исходной датаграммы, принесенный этим фрагментом.

*Заголовок IP-датаграммы*

Поясним названия полей датаграммы $IP_v4$ (рис. 4.6).

*Version* (версия) 4 бита.

Поле версии показывает формат заголовка Internet. Данный документ описывает версию 4.

*IHL* (длина Internet заголовка) 4 бита

Длина Internet заголовка измеряется в словах по 32 бита каждый и указывает на начало поля данных. Заметим, что корректный заголовок может иметь минимальный размер 5 слов.



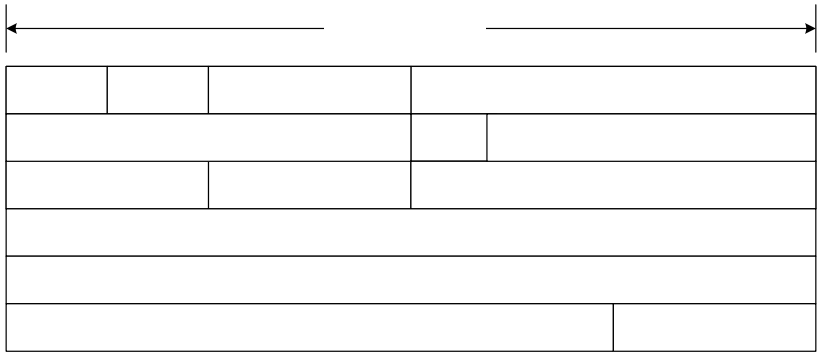


Рисунок 4.6 — Заголовок датаграммы $IP_v4$ в скобках указана длина поля в битах, Options и Padding — поля переменной длины

*Type of Service* (тип сервиса) 8 бит

Тип сервиса определяет с помощью неких абстрактных параметров тип требуемого обслуживания. Эти параметры должны использоваться для управления выбором реальных рабочих характеристик при передаче датаграммы через конкретную сеть. Некоторые сети осуществляют обслуживание с приоритетом, которое неким образом дает преимущество для продвижения данной датаграммы по сравнению со всеми остальными. Реально выбор осуществляется между тремя альтернативами: малой задержкой, высокой достоверностью и высокой пропускной способностью. Биты 0–2 задают уровень приоритета, бит 3 желаемая транзитная задержка (0 — нормальная задержка, 1 — малая задержка); бит 4 желаемая пропускная способность (0 — нормальная пропускная способность, 1 — высокая пропускная способность); бит 5 достоверность (0 — обычная достоверность, 1 — высокая достоверность); биты 6-7 зарезервированы.

*Total Length* (общая длина) 16 бит

Общая длина — это длина датаграммы, измеренная в октетах, включая Internet заголовок и поле данных. Это поле может задавать длину датаграммы вплоть до 65535 октетов.

*Identification* (идентификатор) 16 бит

Идентификатор устанавливается отправителем для сборки фрагментов какой-либо датаграммы.

*Flags* (различные управляющие флаги) 3 бита: бит 0 зарезервирован, должен быть нуль; бит 1 (DF) (0 — возможно фрагментирование, 1 — запрет фрагментации); бит 2 (MF) (0 — последний фрагмент, 1 — будут еще фрагменты).

*TTL — Time to Live* (Время жизни) 8 бит

Это поле показывает максимальное время, в течение которого датаграмме позволено находиться в системе Internet. Если это поле имеет значение нуль, то датаграмма должна быть разрушена. Значение этого поля изменяется при обработке заголовка Internet. Время измеряется в секундах. Однако, поскольку каждый модуль, обрабатывающий датаграмму, должен уменьшать значение поля TTL по крайней мере на единицу, даже если он обрабатывает эту датаграмму менее, чем за секунду, то поле TTL следует понимать как максимальный интервал времени, в течение которого датаграмма может существовать.

*Protocol* (Протокол) 8 бит

Это поле показывает, какой протокол следующего уровня использует данные из Internet датаграммы. Значения для различных протоколов приводятся в документе «Assigned Numbers».

*Header Checksum* (Контрольная сумма заголовка) 16 бит

Поскольку некоторые поля заголовка меняют свое значение (например, время жизни), это значение проверяется и повторно рассчитывается при каждой обработке Internet заголовка. Алгоритм контрольной суммы следующий: поле контрольной суммы — это 16 бит, дополняющие в сумме все 16 битовые слова заголовка. Для вычисления контрольной суммы начальное значение этого поля устанавливается в нуль.

*Source Address* (IP-адрес отправителя) 32 бита (см. п.4.2.1)

*Destination Address* (IP-адрес получателя) 32 бита (см. п. 4.2.1).

*Options* (опции) поле переменной длины.

Опции могут появляться в датаграммах, а могут и не появляться. Они должны поддерживаться всеми Internet модулями (хостами и шлюзами). Не обязательно каждая конкретная датаграмма несет опции, но нести их все же может. В некоторых приложениях опция секретности должна присутствовать во всех датаграммах. Подробно рассматривать опции мы не будем, они достаточно подробго рассмотрены в RFC791.



*Padding* (Выравнивание)

Выравнивание Internet заголовка используется для того, чтобы убедиться в том, что Internet заголовок заканчивается на 32-битной границе. Выравнивание осуществляется нулями. Необходимость в выравнивании существует из-за того, что поле опций — переменной длины.

Следом за заголовком следуют октеты вышестоящих над IP протоколов и данных. Одним из важных протоколов, обеспечивающих надежность работы IP, является ICMP протокол.

### 4.2.4 Протокол контроля сообщений — ICMP

Internet Control Message Protocol (ICMP) описан в документе RFC792.

Иногда шлюз или хост-компьютер, получающий данные, обменивается информацией с хост-компьютером, отправляющим эти данные, то есть в обратном направлении. Именно для таких целей используется данный протокол — протокол контрольных сообщений Internet (ICMP). ICMP использует основные свойства протокола Internet (IP), как если бы ICMP являлся протоколом более высокого уровня. Однако фактически ICMP является составной частью протокола IP.

Сообщения ICMP должны отправляться в некоторых затруднительных ситуациях. Например, когда датаграмма не может достичь своего адресата, когда шлюз не имеет достаточно места в своем буфере для передачи какой-либо датаграммы, или когда шлюз приказывает хост-компьютеру отправлять информацию по более короткому маршруту.

Протокол Internet не создан для того, чтобы обеспечивать абсолютную надежность передачи информации. Целью же данных контрольных сообщений является обеспечение обратной связи, оповещение отправителя данных о проблемах, возникающих в коммуникационном оборудовании. Их целью не является придание надежности протоколу IP. Протокол не дает гарантий, что датаграмма достигает своего адресата или что контрольное сообщение будет возвращено компьютеру, отправившему данные. Некоторые из датаграмм могут исчезнуть в сети, не вызвав при этом никаких оповещений. Протоколы более вы-

сокого уровня, использующие протокол IP, должны применять свои собственные процедуры для обеспечения надежности передачи данных, если таковая требуется.

Виды сообщений, предусмотренные в ICMP, представлены в таблице 4.4.

Таблица 4.4

| Вид сообщения | Описание |
|---|---|
| Порт недостижим | Отправляется, если:<br>1) в таблицах маршрутизации на шлюзах отсутствует путь до адреса назначения;<br>2) датаграмма помечена как нефрагментируемая, но транспортирующая сеть не поддерживает требуемую длину кадра. |
| Превышение контрольного времени | Отправляется, если поле TTL в заголовке IP становится равным 0, но адрес назначения еще не достигнут. |
| Проблемы с параметрами | Если шлюз или хост, обрабатывающий датаграмму, обнаруживает проблему с обработкой параметров заголовка, и это не позволяет завершить ее обработку, то он должен ликвидировать рассматриваемую датаграмму. Одной из потенциальных причин могут быть неправильные аргументы в опции IP. |
| Требование приостановки отправителя | Хост или шлюз исчерпали свои буферы памяти и не успевают обрабатывать поступающие датаграммы. Ситуация «завала». Сообщение о приостановке является запросом для хост-компьютера уменьшить скорость посылки данных на этот конкретный адрес. |
| Сообщение о переадресации | Если шлюз обнаруживает, что хост-компьютер посылает датаграмму неэффективным путем, то шлюз посылает ICMP-сообщение о том, что хост должен использовать другой путь, и указывает этот путь. Хост должен повторить отправку новым маршрутом. |
| Эхо-сообщение и сообщение в ответ на эхо | Используется для диагностики достижимости хоста. Хост-отправитель посылает ICMP сообщение на адрес назначения. Хост получивший данное сообщение, извлекает из него идентификатор и отправляет эхо-сообщение с этим идентификатором. |
| Штамп времени и сообщение с ответом | Штамп времени отправления — это время, которое отправитель фиксировал последний раз перед посылкой сообщения. Штамп времени получения — это время, когда исходное сообщение впервые увидел получатель первоначаль- |



Окончание табл. 4.4

| Вид сообщения | Описание |
|---|---|
| на штамп времени | ного сообщения. Штамп времени передачи — это время, которое фиксировал в последний раз компьютер, отправляющий ответное сообщение. Время — это количество миллисекунд прошедших после наступления текущей даты по единому времени (UT). |
| Запрос информации и ответное сообщение с информацией | Данное сообщение может быть послано, когда в IP заголовке в полях отправителя и получателя записаны нули (это означает «именно эту» локальную сеть). В ответ должен быть послан IP модуль с полностью заданными адресами. Данное сообщение является способом, с помощью которого хост-компьютер сможет определить номер сети, куда он подключен. |

Механизм эхо-сообщений используется в такой сетевой утилите командной строки как ping.

Утилита tracert (англ. trace route — трассировка маршрута) позволяет отследить все шлюзы, через которые проходит пакет на пути к адресату. Для этого используется сообщение ICMP о превышении контрольного времени (рис. 4.7). На первой итерации хост-отправитель A посылает IP-датаграмму хосту-получателю B с установленным значением поля TTL = 1. Как только датаграмма попадает на первый шлюз, значение TTL уменьшается на единицу и становится равным 0.

Шлюз, обнаружив, что время существования датаграммы истекло, разрушает эту датаграмму, генерирует соответствующие ICMP-сообщение и посылает его отправителю. Так, из полученного ICMP-сообщения, хост-отправитель узнает адрес первого шлюза G1 на пути до адресата. Обратный путь ICMP-сообщения на рис. 4.7 показан пунктиром потому, что фактически это сообщение проходит те же шлюзы, что и исходная датаграмма.

На второй итерации хост-отправитель формирует IP-датаграмму с установленным значением поля TTL = 2. Теперь первый шлюз, уменьшив значение на 1, получит TTL равное 1.

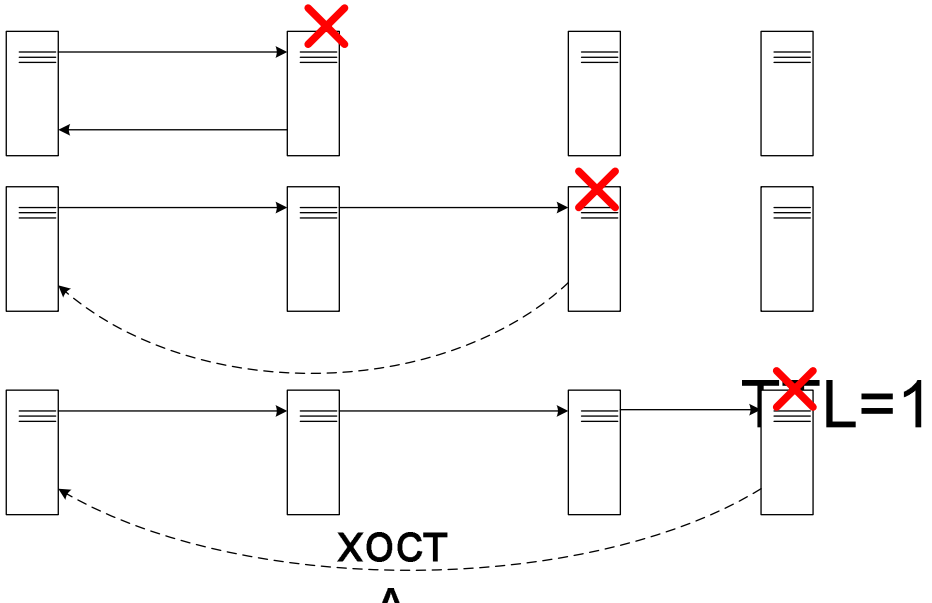



Рисунок 4.7 — Принцип работы утилиты трассировки маршрута tracert

Контрольное время не истекло, значит датаграмму можно пересылать дальше. Второй шлюз G2 (или маршрутизатор), в свою очередь уменьшит значение поля на 1, и, получив 0, сформирует ICMP-сообщение и перешлет его назад, на адрес отправителя. Если ничего непредвиденного не произойдет, то это сообщение достигнет хоста-отправителя, а так как при отправке ICMP второй шлюз G2 подставил в качестве адреса отправителя свой адрес, то хост отправитель узнает, какой шлюз является вторым на пути к адресу назначения.

На третьей итерации все произойдет аналогично, только TTL будет равно 3. На этот раз датаграмма достигнет хоста-получателя, благополучно уничтожится, утилита tracert на хосте-отправителе получит адрес последнего хоста, и посылка датаграмм прекратится.



### 4.2.5 Протокол датаграмм пользователя — UDP

User Datagram Protocol (RFC768) является протоколом более высокого уровня по отношению к IP. Данный протокол предоставляет прикладной программе процедуру для посылки сообщений другим программам, причем механизм протокола минимален. Протокол UDP ориентирован на транзакции, то есть для передачи коротких, независимых друг от друга сообщений. Получение датаграмм и защита от дублирования не гарантированы. Приложения, требующие гарантированного получения потоков данных, должны использовать протокол управления пересылкой (Transmission Control Protocol — TCP).

В то время как протокол IP является скрытым от клиентского программного обеспечения, UDP полностью доступен прикладному ПО. Если клиентское программное обеспечение «желает» пересылать данные в виде независимых датаграмм, то UDP представляет собой упрощенную форму IP. Формируя заголовок UDP-датаграммы, клиент должен определить только порт отправителя, порт получателя и длину датаграммы (рис. 4.8 ). При пересылке, UDP упаковывается внутрь IP датаграммы. Для клиентского программного обеспечения это происходит незаметно.

Одним из высокоуровневых протоколов, использующих UDP, является Trivial File Transfer Protocol (TFTP) — тривиальный протокол передачи файлов.

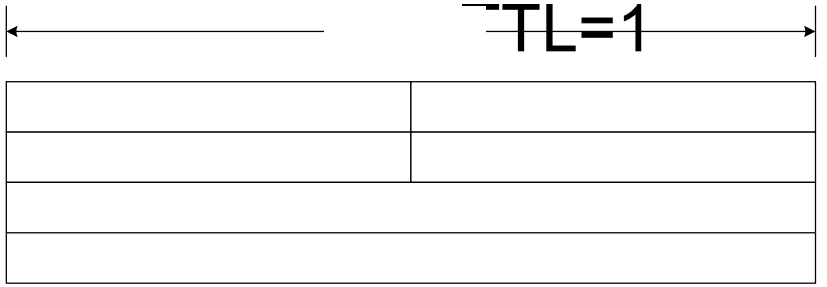


Рисунок 4.8 — Формат UDP-датаграммы

### 4.2.6 Протокол контроля передачи — TCP

Transmission Control Protocol (RFC793) предназначен для использования в качестве надежного протокола общения между хост-компьютерами в коммуникационных компьютерных сетях с коммутацией пакетов, а также в системах, объединяющих такие сети. TCP занимает в многоуровневой архитектуре протоколов нишу непосредственно над протоколом IP, который позволяет протоколу TCP отправлять и получать сегменты информации переменной длины, заключенные в оболочку IP-датаграмм. IP-датаграмма предоставляет средства для адресации отправителя и получателя сегментов TCP в различных сетях. IP также осуществляет любую фрагментацию и сборку сегментов TCP, необходимую для осуществления передачи и доставки через множество сетей и промежуточных шлюзов.

Главной целью протокола TCP является обеспечение надежного, безопасного сервиса для логических цепей или соединений между парами процессов. Чтобы обеспечить такой сервис, основываясь на менее надежных коммуникациях Internet, система должна иметь возможности для работы в следующих областях:

- базовая передача данных;
- достоверность;
- управление потоком;
- разделение каналов;
- работа с соединениями;
- приоритет и безопасность.

*Базовая передача данных*

Протокол TCP способен передавать непрерывные потоки октетов между своими клиентами в обоих направлениях, пакуя некое количество октетов в сегменты для передачи через системы Internet. В общем случае протоколы TCP решают по своему усмотрению, когда производить блокировку и передачу данных.

Иногда пользователям бывает необходимо убедиться в том, что все данные, переданные ими протоколу TCP, уже отправлены. Для этой цели определена функция проталкивания (push). Чтобы убедиться в том, что данные, отправленные протоколу



TCP, действительно переданы, отправитель указывает, что их следует протолкнуть к получателю. Проталкивание приводит к тому, что программы протокола TCP сразу осуществляют отправление и, соответственно, получение остающихся данных. Правильно осуществленное проталкивание может быть невидимо для получателя, а сама функция проталкивания может не иметь маркера границы записи.

*Достоверность*

Протокол TCP должен иметь защиту от разрушения данных, потери, дублирования и нарушения очередности получения, вызываемых коммуникационной системой Internet. Это достигается присвоением очередного номера каждому передаваемому октету, а также требованием подтверждения (ACK) от программы TCP, принимающей данные. Если подтверждения не получено в течение контрольного интервала времени, то данные посылаются повторно. Со стороны получателя номера очереди используются для восстановления очередности сегментов, которые могут быть получены в неправильном порядке, а также для ограничения возможности появления дубликатов. Повреждения фиксируются посредством добавления к каждому передаваемому сегменту контрольной суммы, проверки ее при получении и последующей ликвидации дефектных сегментов. До тех пор, пока программы протокола TCP продолжают функционировать корректно, а система Internet не развалилась полностью на составные части, ошибки пересылки не будут влиять на правильное получение данных.

*Управление потоком*

Протокол TCP дает средства получателю управлять количеством данных, посылаемых ему отправителем. Это достигается возвратом так называемого «окна» (window) вместе с каждым подтверждением, которое указывает диапазон приемлемых номеров, следующих за номером последнего успешно принятого сегмента. Окно определяет количество октетов, которое отправитель может послать до получения дальнейших указаний.

*Разделение каналов*

Чтобы позволить на отдельно взятом компьютере многим процессам одновременно использовать коммуникационные возможности уровня TCP, протокол TCP предоставляет на каждом хост-компьютере набор адресов или портов. Вместе с адресами сетей и хост-компьютеров на коммуникационном уровне Internet они образуют сокет (socket — разъем).

Каждое соединение уникальным образом идентифицируется парой сокетов. Таким образом, любой сокет может одновременно использоваться во многих соединениях.

Соотнесение портов и процессов осуществляется каждым хост-компьютером самостоятельно.

Механизмы управления потоком и обеспечения достоверности, описанные выше, требуют, чтобы программы протокола TCP инициализировали и поддерживали определенную информацию о состоянии каждого потока данных. Набор такой информации, включающий сокеты, номера очереди, размеры окон, называется соединением. Каждое соединение уникальным образом идентифицируется парой сокетов на двух концах.

Если два процесса желают обмениваться информацией, соответствующие программы протокола TCP должны сначала установить соединение (на каждой стороне инициализировать информацию о статусе). По завершении обмена информацией соединение должно быть расторгнуто или закрыто, чтобы освободить ресурсы для предоставления другим пользователям.

Поскольку соединения должны устанавливаться между ненадежными хост-компьютерами и через ненадежную коммуникационную систему Internet, то во избежание ошибочной инициализации соединений используется механизм подтверждения связи с хронометрированными номерами очереди.

*Приоритет и безопасность*

Пользователи протокола TCP могут затребовать для своего соединения приоритет и безопасность. Предусмотрены принимаемые по умолчанию характеристики соединений, когда такие параметры не требуются.



*Принцип действия TCP*

Ключевым свойством TCP, определяющим всю структуру протокола, является то, что в TCP-соединении у каждого байта есть свой 32-разрядный порядковый номер. Отдельные 32-разрядные порядковые номера используются для подтверждений и для механизма скользящего окна, о чем также будет сказано позже. Отправляющая и принимающая TCP-сущности (программы, использующие TCP соединение) обмениваются данными в виде сегментов. Сегмент состоит из фиксированного 20-байтового заголовка (плюс необязательная часть), за которой могут следовать байты данных. Размер сегментов определяется программным обеспечением TCP. Оно может объединять в один сегмент данные, полученные в результате нескольких операций записи, или, наоборот, распределять результат одной записи между несколькими сегментами. Размер сегментов ограничен двумя пределами. Во-первых, каждый сегмент, включая TCP-заголовок, должен помещаться в 65 515-байтное поле полезной нагрузки IP-пакета. Во-вторых, в каждой сети есть максимальная единица передачи (MTU, Maximum Transfer Unit), и каждый сегмент должен помещаться в MTU. На практике размер максимальной единицы передачи составляет обычно 1500 байт (что соответствует размеру поля полезной нагрузки Ethernet), и таким образом определяется верхний предел размера сегмента.

Основным протоколом, используемым TCP-сущностями, является протокол скользящего окна. При передаче сегмента отправитель включает таймер. Когда сегмент прибывает в пункт назначения, принимающая TCP-сущность посылает обратно сегмент (с данными, если есть, что посылать, или без данных) с номером подтверждения, равным порядковому номеру следующего ожидаемого сегмента. Если время ожидания подтверждения истекает, отправитель посылает сегмент еще раз.

Хотя этот протокол кажется простым, в нем имеется несколько деталей, которые следует рассмотреть подробнее. Сегменты могут приходить в неверном порядке. Так, например, возможна ситуация, в которой байты с 3072-го по 4095-й уже прибыли, но подтверждение для них не может быть выслано, так как байты с 2048-го по 3071-й еще не получены. К тому же сегменты могут задерживаться в сети так долго, что у отправи-

теля истечет время ожидания и он передаст их снова. Переданный повторно сегмент может включать в себя уже другие диапазоны фрагментов, поэтому потребуется очень аккуратное администрирование для определения номеров байтов, которые уже были приняты корректно. Поскольку каждый байт в потоке единственным образом определяется по своему сдвигу, эта задача оказывается реальной.

Протокол TCP должен уметь справляться с этими проблемами и решать их эффективно. На оптимизацию производительности TCP-потоков было потрачено много сил.

Рассмотрим структуру заголовка TCP-сегмента (рис. 4.9). Каждый сегмент начинается с 20-байтного заголовка фиксированного формата. За ним могут следовать дополнительные поля. После дополнительных полей может располагаться до 65 535 – – 20 – 20 = 65 495 байт данных, где первые 20 байт — это IP-заголовок, а вторые — TCP-заголовок. Сегменты могут и не содержать данных. Такие сегменты часто применяются для передачи подтверждений и управляющих сообщений.

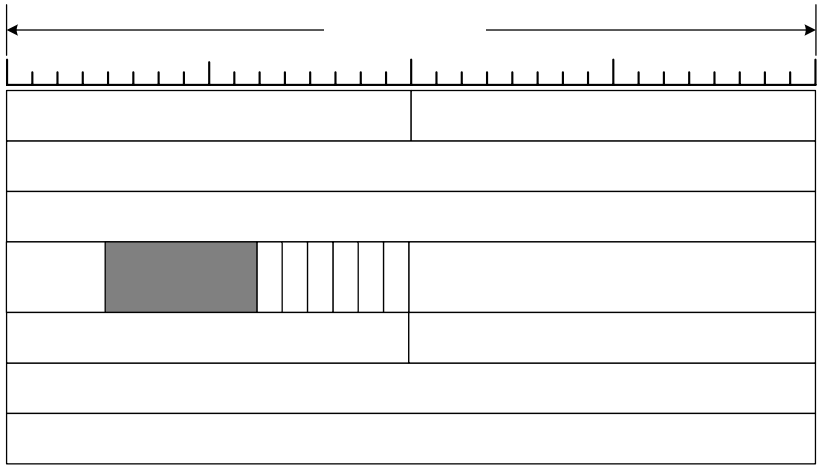


Рисунок 4.9 — Состав и структура TCP-сегмента

Рассмотрим TCP-заголовок поле за полем. Поля *Порт получателя* и *Порт отправителя* являются идентификаторами



локальных конечных точек соединения. Популярные номера портов перечислены на www.iana.org, однако, что касается всех остальных портов, то каждый хост может сам решать, как их распределять. Номер порта вместе с IP-адресом хоста образуют уникальный 48-битный идентификатор конечной точки. Пара таких идентификаторов, относящихся к источнику и приемнику, однозначно определяет соединение.

Поля *Порядковый номер* и *Номер подтверждения* выполняют свою обычную функцию. Обратите внимание: поле *Номер подтверждения* относится к следующему ожидаемому байту, а не к последнему полученному. Оба они 32-разрядные, так как в TCP-потоке нумеруется каждый байт данных.

Поле *Длина TCP-заголовка* содержит размер TCP-заголовка, выраженный в 32-разрядных словах. Эта информация необходима, так как поле *Факультативные поля,* а вместе с ним и весь заголовок, может быть переменной длины. По сути, это поле указывает смещение от начала сегмента до поля данных, измеренное в 32-битных словах. Это то же самое, что длина заголовка.

Следом идет неиспользуемое 6-битное поле. Тот факт, что это поле выжило в течение четверти века, является свидетельством того, насколько хорошо продуман дизайн TCP.

Затем следуют шесть 1-битовых флагов. Бит *URG* устанавливается в единицу в случае использования поля *Указатель на срочные данные,* содержащего смещение в байтах от текущего порядкового номера байта до места расположения срочных данных. Таким образом, в протоколе TCP реализуются прерывающие сообщения. Как уже упоминалось, протокол TCP лишь обеспечивает доставку сигнала пользователя до получателя, не интересуясь причиной прерывания.

Если бит *ACK* установлен в единицу, значит, поле *Номер подтверждения* содержит осмысленные данные. В противном случае данный сегмент не содержит подтверждения, и поле *Номер подтверждения* просто игнорируется.

Бит *PSH* является, по сути, PUSH-флагом, с помощью которого отправитель просит получателя доставить данные приложению сразу по получении пакета, а не хранить их в буфере, пока тот не наполнится. Получатель может заниматься буферизацией для достижения большей эффективности.

Бит *RST* используется для сброса состояния соединения, которое из-за сбоя хоста или по другой причине попало в тупиковую ситуацию. Кроме того, он используется для отказа от неверного сегмента или от попытки создать соединение. Если вы получили сегмент с установленным битом *RST,* это означает наличие какой-то проблемы.

Бит *SYN* применяется для установки соединения. У запроса соединения бит *SYN*= 1, а бит *ACK=0,* что означает, что поле подтверждения не используется. В ответе на этот запрос содержится подтверждение, поэтому значения этих битов в нем равны: *SYN*= 1, *ACK*= 1. Таким образом, бит *SYN* используется для обозначения сегментов CONNECTION REQUEST и CONNECTION ACCEPTED, а бит *ACK* — чтобы отличать их друг от друга.

Бит *FIN* используется для разрыва соединения. Он указывает на то, что у отправителя больше нет данных для передачи. Однако, даже закрыв соединение, процесс может продолжать получать данные в течение неопределенного времени. У сегментов с битами *FIN* и *SYN* есть порядковые номера, что гарантирует правильный порядок их выполнения.

Управление потоком в протоколе TCP осуществляется при помощи скользящего окна переменного размера. Поле *Размер окна* сообщает, сколько байт может быть послано после байта, получившего подтверждение. Значение поля *Размер окна* может быть равно нулю, что означает, что все байты вплоть до *Номер подтверждения* —получены, но у получателя в данный момент какие-то проблемы, и остальные байты он пока принять не может. Разрешение на дальнейшую передачу может быть получено путем отправки сегмента с таким же значением поля *Номер подтверждения* и ненулевым значением поля *Размер окна.*

В TCP подтверждения отделены от разрешений на передачу данных. В сущности, приемник может сказать: «Я получил байты вплоть до *k*-го*,* но я сейчас не хочу продолжать прием данных». Такое разделение (выражающееся в скользящем окне *переменного размера)* придает протоколу дополнительную гибкость.

Поле *Контрольная сумма* служит для повышения надежности. Оно содержит контрольную сумму заголовка, данных и



псевдозаголовка (рис. 4.10). При выполнении вычислений поле *Контрольная сумма* устанавливается равным нулю, а поле данных дополняется нулевым байтом, если его длина представляет собой нечетное число. Алгоритм вычисления контрольной суммы просто складывает все 16-разрядные слова в дополнительном коде, а затем вычисляет дополнение для всей суммы. В результате, когда получатель считает контрольную сумму всего сегмента, включая поле *Контрольная сумма,* результат должен быть равен 0.


Рисунок 4.10 — Псевдозаголовок, включаемый в контрольную сумму TCP

Псевдозаголовок содержит 32-разрядные IP-адреса отправителя и получателя, номер протокола для TCP (6- в настоящее время) и счетчик байтов для TCP-сегмента (включая заголовок). Включение псевдозаголовка в контрольную сумму TCP помогает обнаружить неверно доставленные пакеты, хотя это нарушает иерархию протоколов, так как IP-адреса в нем принадлежат IP-уровню, а не TCP-уровню. В UDP для контрольной суммы используется такой же псевдозаголовок.

Поле *Факультативные поля* предоставляет дополнительные возможности, не покрываемые стандартным заголовком. С помощью одного из таких полей каждый хост может указать максимальный размер поля полезной нагрузки, который он может принять. Чем больше размер используемых сегментов, тем выше эффективность, так как при этом снижается удельный вес накладных расходов в виде 20-байтных заголовков, однако не все хосты способны принимать очень большие сегменты. Хосты могут сообщить друг другу максимальный размер поля полезной нагрузки во время установки соединения. По умолчанию

этот размер равен 536 байтам. Все хосты обязаны принимать TCP-сегменты размером 536 + 20 = 556 байт. Для каждого из направлений может быть установлен свой максимальный размер поля полезной нагрузки.

Для линий с большой скоростью передачи и/или большой задержкой окно размером в 64 Кбайт оказывается слишком маленьким. Так, для линии Т3 (44,736 Мбит/с) полное окно может быть передано в линию всего за 12 мс. Если значение времени распространения сигнала в оба конца составляет 50 мс (что типично для трансконтинентального оптического кабеля), 3/4 времени отправитель будет заниматься ожиданием подтверждения. При связи через спутник ситуация будет еще хуже. Больший размер окна мог бы улучшить эффективность, но 16-битовое поле *Размер окна* не позволяет этого сделать. В RFC 1323 был предложен новый параметр *Масштаб окна,* о значении которого два хоста могли договориться при установке соединения. Это число позволяет сдвигать поле *Размер окна* до 14 разрядов влево, обеспечивая расширение размера окна до $2^{30}$ байт (1 Гбайт). В настоящее время большинство реализаций протокола TCP поддерживают эту возможность.

Еще одна возможность, предложенная в RFC 1106 и широко применяемая сейчас, состоит в использовании протокола выборочного повтора вместо возврата на *n.* Если адресат получает один плохой сегмент и следом за ним большое количество хороших, у нормального TCP-протокола в конце концов истечет время ожидания и он передаст повторно все неподтвержденные сегменты, включая те, что были получены правильно. В документе RFC 1106 было предложено использовать отрицательные подтверждения (NAK), позволяющие получателю запрашивать отдельный сегмент или несколько сегментов. Получив его, принимающая сторона может подтвердить все хранящиеся в буфере данные, уменьшая, таким образом, количество повторно передаваемых данных.

*Модель управления TCP-соединением*

Этапы, необходимые для установки и разрыва соединения, могут быть представлены в виде модели конечного автомата (рис. 4.11), 11 состояний которого перечислены в таблице 4.5.



В каждом из этих состояний могут происходить разрешенные и запрещенные события.

Таблица 4.5

| Состояние | Описание |
|---|---|
| CLOSED | Закрыто. Соединение не является активным и не находится в процессе установки |
| LISTEN | Ожидание. Сервер ожидает входящего запроса |
| SYN RCVD | Прибыл запрос соединения. Ожидание подтверждения |
| SYN SENT | Запрос соединения послан. Приложение начало открывать соединение |
| ESTABLISHED | Установлено. Нормальное состояние передачи данных |
| FIN WAIT 1 | Приложение сообщило, что больше нечего передавать |
| FIN WAIT 2 | Другая сторона согласна разорвать соединение |
| TIMED WAIT | Ожидание, пока в сети не исчезнут все пакеты |
| CLOSING | Обе стороны попытались одновременно закрыть соединение |
| CLOSE WAIT | Другая сторона инициировала разъединение |
| LAST ACK | Ожидание, пока в сети не исчезнут все пакеты |

В ответ на какое-либо разрешенное событие может осуществляться определенное действие. При возникновении запрещенных событий сообщается об ошибке.

Каждое соединение начинается в состоянии *CLOSED* (закрытое). Оно может выйти из этого состояния, предпринимая либо активную (CONNECT), либо пассивную (LISTEN) попытку открыть соединение. Если противоположная сторона осуществляет противоположные действия, соединение устанавливается и переходит в состояние *ESTABLISHED.* Инициатором разрыва соединения может выступить любая сторона. По завершении процесса разъединения соединение возвращается в состояние *CLOSED.*

Типичный случай клиента, активно соединяющегося с пассивным сервером, показан жирными линиями — сплошными для клиента и пунктирными для сервера. Тонкие линии обозначают необычные последовательности событий. Каждая линия на рис. 4.11 маркирована парой *событие/действие.* Событие может представлять собой либо обращение пользователя к системной

процедуре (CONNECT, LISTEN, SEND или CLOSE), либо прибытие сегмента *(SYN, FIN, ACK* или *RST),* либо, в одном случае, окончание периода ожидания, равного двойному времени жизни пакетов. Действие может состоять в отправке управляющего сегмента *(SYN, FIN* или *RST).* Впрочем, может не предприниматься никакого действия, что обозначается прочерком. В скобках приводятся комментарии.

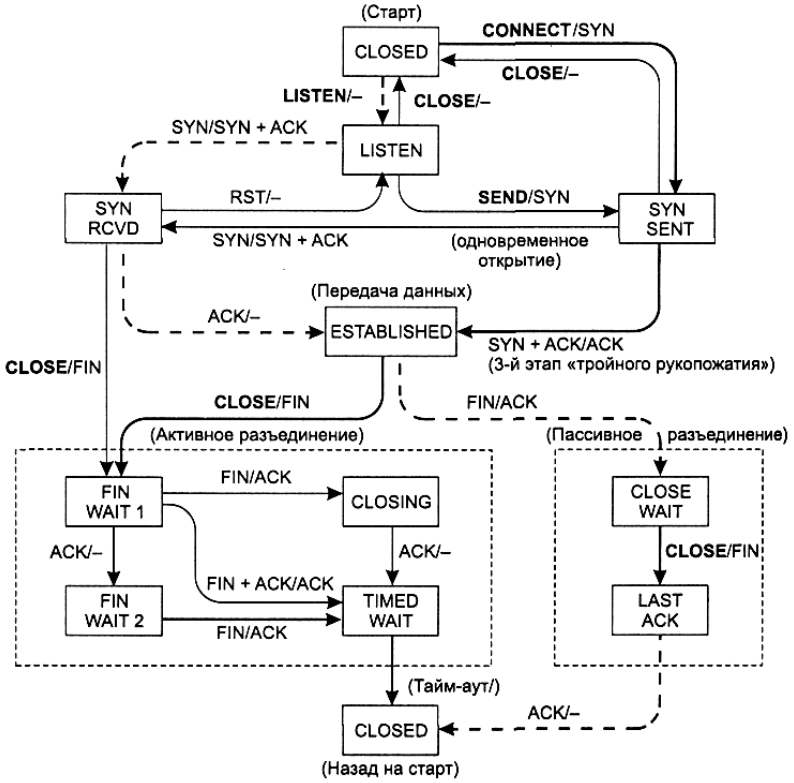



Рисунок 4.11 — Конечный автомат TCP-соединения. Жирная сплошная линия показывает нормальный путь клиента. Пунктиром показан нормальный путь сервера. Тонкими линиями обозначены необычные события



Диаграмму легче всего понять, если сначала проследовать по пути клиента (сплошная жирная линия), а затем — по пути сервера (жирный пунктир). Когда приложение на машине клиента вызывает примитив CONNECT, локальная TCP-сущность создает запись соединения, помечает его состояние как *SYN SENT* и посылает *SYN-сегмент.* Примечательно, что несколько приложений одновременно могут открыть несколько соединений, поэтому свое состояние, хранящееся в записи соединения, имеется у каждого отдельного соединения. Когда прибывает сегмент *SYN + ACK,* TCP-сущность посылает последний *ACK-сегмент* «тройного рукопожатия» и переключается в состояние *ESTABLISHED.* В этом состоянии можно пересылать и получать данные.

Когда у приложения заканчиваются данные для передачи, оно выполняет примитив CLOSE, заставляющий локальную TCP-сущность послать *FIN-сегмент* и ждать ответного *ACK*-сегмента (пунктирный прямоугольник с пометкой «активное разъединение»). Когда прибывает подтверждение, происходит переход в состояние *FIN WAIT 2,* и одно направление соединения закрывается. Когда приходит встречный *FIN*-сегмент, в ответ на него также высылается подтверждение, и второе направление соединения также закрывается. Теперь обе стороны соединения закрыты, но TCP-сущность выжидает в течение времени, равного максимальному времени жизни пакета, чтобы можно было гарантировать, что все пакеты этого соединения больше не перемещаются по сети даже в том случае, если подтверждение было потеряно. Когда этот период ожидания истекает, TCP-сущность удаляет запись о соединении.

Рассмотрим теперь управление соединением с точки зрения сервера. Сервер выполняет примитив LISTEN и переходит в режим ожидания запросов соединения. Когда приходит *SYN*-сегмент, в ответ на него высылается подтверждение, после чего сервер переходит в состояние *SYN RCVD* (запрос соединения получен). Когда в ответ на *SYN*-подтверждение сервера от клиента приходит *ACK*-сегмент, процедура «тройного рукопожатия» завершается и сервер переходит в состояние *ESTABLISHED.* Теперь можно пересылать данные.

По окончании выполнения своей задачи клиент запускает примитив CLOSE, в результате чего на сервер прибывает *FIN*-сегмент (пунктирный прямоугольник, обозначенный как пассивное разъединение). Теперь сервер выполняет примитив CLOSE, а *FIN*-сегмент посылается клиенту. Когда от клиента прибывает подтверждение, сервер разрывает соединение и удаляет запись о нем.

## 4.3 Программирование сокетов Беркли

WinSock Application Programming Interface (API) — это интерфейс взаимодействия со стеками коммуникационных протоколов в операционных системах семейства Windows. Большинство современных версий поддерживают только стек TCP/IP.

Предполагается, что будущие версии будут поддерживать также и IPX/SPX.

Современная реализация WinSock API поддерживает основные возможности Berkely Sockets, достаточные для создания эффективных сетевых приложений.

### 4.3.1 Примитивы сокетов Беркли

Набор сокетов Беркли несколько не соответствует стандартному набору примитивов транспортного уровня ВОС (таблица 4.6). Хотя основное назначение такое же.

Таблица 4.6

| Примитив | Назначение |
|---|---|
| SOCKET (гнездо) | Дескриптор сокета, именование ресурса |
| BIND (связать) | Связывает ресурсы (сокет) с локальным адресом хоста |
| LISTEN (ожидать) | Возможность принять данные с указанием размера очереди |
| ACCEPT (принять) | Блокирует сервер до попытки соединения клиента |
| CONNECT (соединить) | Попытка установить соединения |
| SEND (переслать) | Посылает данные по соединению |
| RECEIVE (получить) | Получает данные у соединения |
| CLOSE (закрыть) | Разрывает соединение |



Сокет обычно трактуется как дескриптор файла и указывается в качестве параметра практически во всех примитивах.

Рассмотрим, как используются примитивы на стороне сервера и на стороне клиента.

### 4.3.2 Сервер

Вызовы примитивов осуществляются в следующем порядке:

SOCKET — создает дескриптор (примерно так же как OPEN при открытии файла) и выделяет для него место в таблице транспортного объекта. При этом сообщается, какая служба необходима, канальная (TCP) или дейтаграммная (UDP).

BIND — привязывает конкретный сетевой адрес к сокету, после чего становится возможной подключение клиентов.

LISTEN — выделяет место для очереди входящих вызовов на случай одновременного обращения нескольких клиентов.

ACCEPT — сервер переходит в режим ожидания (блокируется) до прихода блока с запросом соединения от клиента. При этом создается новый сокет, с теми же параметрами, что и оригинального сокета. Сервер может распараллелить процесс обработки нового сокета, чтобы иметь возможность вернуться к ожиданию новых соединений.

Ниже в листинге 4.2 приведен пример сервера, который передает клиенту содержимое файла в ответ на присланное клиентом имя файла. Заголовочные файлы приведены для системы UNIX. Для Windows используются другие заголовочные файлы, а также используются дополнительные функции: WSAstartup и др. Исходный код сервера приведен на компакт-диске.

Листинг 4.2 Сервер, передающий клиенту файл с заданным именем

```
#include <sys/types.h>
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 12345  /* установка порта */
#define BUF_SIZE 4096      /* размер буфера передачи */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
```

```
int s, b, l, fd, sa, bytes, on = 1;
char buf[BUF_SIZE];/* буфер для исходящего файла */
struct sockaddr_in channel;     /* структура для адреса, используется в IP-заголовке */
/* Строим адресную структуру для привязки к сокету */
memset(&channel, 0, sizeof(channel));  /* нулевой канал */
channel.sin_family = AF_INET;
channel.sin_addr.s_addr = htonl(INADDR_ANY); /* записываем адрес в сетевом порядке байт*/
channel.sin_port = htons(SERVER_PORT); /* записываем порт в сетевом порядке байт*/
/* Ждем соединения */
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); /* создаем сокет*/
if (s < 0) fatal("socket failed"); /*проверка на ошибку при создании сокета*/
setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on)); /*устанавливаем опции сокета*/
b = bind(s, (struct sockaddr *) &channel, sizeof(channel)); /*связывает сокет с конкретным сетевым адресом*/
if (b < 0) fatal("bind failed"); /*проверка на ошибку при выполнении bind*/
l = listen(s, QUEUE_SIZE);     /*ставим сокет на прослушивание, сообщаем размер очереди */
if (l < 0) fatal("listen failed"); /*проверка на ошибку при выполнении listen*/
/* Сокет создан, ждем в бесконечном цикле соединений для обработки */
while (1) {
    sa = accept(s, 0, 0); /*блокировка соединения, ждем клиента*/
    if (sa < 0) fatal("accept failed"); /*проверка на ошибку accept*/
    read(sa, buf, BUF_SIZE); /*читаем из сокета имя файла в буфер*/
  /*Открываем файл на чтение с указанным именем  */
   fd = open(buf, O_RDONLY);
   if (fd < 0) fatal("open failed"); /*а вдруг такого файла нет? */
/*Бесконечный цикл для получения содержимого файла из сети, выход из цикла осуществляется оператором break, если в очередной прием в буфере оказалось 0 байт*/
     while (1) {
          bytes = read(fd, buf, BUF_SIZE); /* читаем из файла сегмент в буфер */
          if (bytes <= 0) break;  /*если конец файла, выходим из цикла чтения */
          write(sa, buf, bytes);    /* пишем очередной сегмент файла в сокет, что равносильно отправке в сеть*/
     }
     close(fd);              /* закрываем файл */
```



```
     close(sa);              /* закрываем соединение */
  }
}
/* описание функции fatal, основное назначение вывести сообщение на экран и завершить работу программы с кодом 1*/
fatal(char *string)
{
  printf("%s", string);
  exit(1);
}
```

### 4.3.3 Клиент

На стороне клиента используются следующие примитивы:

1. SOCKET — используется аналогично серверу, для обозначения ресурсов, выделенных для данного соединения;
2. CONNECT — блокирует вызывающего до завершения процесса фазы установки соединения, то есть до подтверждения сервера, что соединение установлено;
3. после установления соединения, между сервером и клиентом идет обмен с использованием RECEIVE и SEND;
4. соединение разрывается, если обе стороны используют примитив CLOSE.

После запуска на любом узле сети Интернет, приложение-клиент может приложению-серверу имя файла и получить содержимое этого файла. Клиентская часть тандема приведена в листинге 4.3  в версии для UNIX. Пример кода клиента для Windows приведен в проекте winsocket на прилагающемся компакт диске.

Листинг 4.3 Клиент, запрашивающий у сервера содержимое файла с определенным именем

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 12345   /* установка порта, номер должен совпадать с сервером */
#define BUF_SIZE 4096   /* размер буфера передачи */
/*Функция main, в качестве аргументов командной строки получает имя сервера и имя файла, который должен скопироваться */
int main(int argc, char **argv)
{
```

```
        int c, s, bytes;
        char buf[BUF_SIZE];     /* буфер для входящего файла */
        struct hostent *h;      /* информация о сервере */
        struct sockaddr_in channel; /* структура для адреса */

        if (argc != 3) fatal("Usage: client server-name file-name");
       /*возвращаем ошибку, если в командной строке не сообщены
имя сервера и/или имя клиента*/
        h = gethostbyname(argv[1]);      /* получение IP по адресу
сервера */
        if (!h) fatal("gethostbyname failed");
        s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (s < 0) fatal("socket");
        memset(&channel, 0, sizeof(channel));
        channel.sin_family= AF_INET;
        memcpy(&channel.sin_addr.s_addr, h->h_addr, h-
>h_length);
        channel.sin_port= htons(SERVER_PORT);
        c = connect(s, (struct sockaddr *) &channel,
sizeof(channel));
        if (c < 0) fatal("connect failed");
        /* Соединение установлено пересылка имени файла, вклю-
чая нулевой сисмвол */
        write(s, argv[2], strlen(argv[2])+1);

        /* Получение файла и отправка на стандартный вывод */
        while (1) {
            bytes = read(s, buf, BUF_SIZE); /*читать из socket */
            if (bytes <= 0) exit(0);          /* конец файла? */
            write(1, buf, bytes);             /* писать в output */
        }
       }
       /*Функция вывода сообщения об ошибке и завершение рабо-
ты программы*/
       fatal(char *string)
       {
        printf("%s\n", string);
        exit(1);
       }
```



# 5 ПРИКЛАДНОЙ УРОВЕНЬ СТЕКА TCP/IP

## 5.1 Электронная почта

Электронная почта, или e-mail**,** как называют ее многочисленные любители, существует уже более двух десятилетий. До 1990 года она использовалась преимущественно в научных организациях. В 90-е годы она получила широкую известность, и с тех пор количество отправляемых с помощью электронной почты писем стало расти экспоненциально. Среднее число сообщений, посылаемых ежедневно, скоро во много раз превзошло число писем, отправляемых с помощью обычной, бумажной почты.

Электронной почте, как и любой форме коммуникаций, присущ определенный стиль и набор соглашений. В частности, общение по электронной почте носит очень неформальный и демократичный характер. Скажем, человек, который никогда бы не осмелился позвонить или даже написать бумажное письмо какой-нибудь «Особо Важной Персоне», запросто может сесть и написать ей небрежное электронное сообщение.

Первые системы электронной почты состояли просто из протоколов передачи файлов и договоренности указывать адрес получателя в первой строке каждого сообщения (то есть файла). Со временем недостатки данного метода стали очевидны. Перечислим некоторые из них:

1. Было очень неудобно отсылать сообщения группе получателей. Эта возможность часто требовалась менеджерам для рассылки уведомлений своим подчиненным.

2. Сообщения не обладали внутренней структурой, что затрудняло их компьютерную обработку. Например, если переадресованное сообщение было помещено в тело другого сообщения, извлечь одно сообщение из другого было довольно сложно.

3. Отправитель сообщения никогда не знал, дошло ли его сообщение до адресата.

4. Если кто-либо собирался уехать на несколько недель по делам и хотел, чтобы вся его почта переправлялась его секретарю, организовать это было непросто.

5. Интерфейс пользователя практически отсутствовал. Пользователь должен был сначала в текстовом редакторе набрать сообщение, затем выйти из редактора и запустить программу передачи файла.

6. Было невозможно создавать и посылать сообщения, содержащие смесь текста, изображений (рисунков, факсов, фото) и звука.

Со временем, когда накопился опыт работы, были предложены более сложные системы электронной почты. В 1982 году предложения по работе с электронной почтой, выдвинутые администрацией сети ARPANET, были опубликованы в виде RFC 821 (протокол передачи) и RFC 822 (формат сообщений). Подвергшись минимальным изменениям, нашедшим отражение в RFC 2821 и 2822, они фактически стали стандартами Интернета, однако все равно, говоря об электронной почте Интернета, многие ссылаются на RFC 822.

В 1984 году консультативный комитет по международной телефонии и телеграфу (CCITT) впервые представил стандарт X.400. После двух десятков лет борьбы электронная почта, основанная на стандарте RFC 822, получила широкое применение, тогда как системы, базирующиеся на стандарте X.400, практически исчезли. Получилось так, что система, созданная горсткой аспирантов-компьютерщиков, смогла превзойти официальный международный стандарт, имевший серьезную поддержку всех управлений почтово-телеграфной и телефонной связи во всем мире, многих правительств и значительной части компьютерной промышленности. Причина успеха стандарта RFC 822 кроется не столько в его достоинствах, сколько в недостатках стандарта X.400. Последний был настолько плохо продуман и так сложен, что никто просто не мог его нормально реализовать. Большинство организаций предпочли простую, но работающую систему электронной почты, основанную на RFC 822, возможно, действительно замечательной, но неработающей системе, в основе которой был стандарт X.400.

*Архитектура и службы*

В данном разделе мы рассмотрим возможности и организацию систем электронной почты. Обычно они состоят из двух подсистем: пользовательских агентов, позволяющих пользователям читать и отправлять электронную почту, и агентов передачи сообщений, пересылающих сообщения от отправителя к получателю. Пользовательские агенты представляют собой локальные программы, предоставляющие различные методы взаимодействия пользователя с почтовой системой. Эти методы (или интерфейсы) могут быть командными, графическими или основанными на меню. Агенты передачи сообщений обычно являются системными демонами — процессами, работающими в фоновом режиме, и перемещающими электронную почту по системе.



Обычно системами электронной почты поддерживаются следующие пять основных функций.

*Составление* — процесс создания сообщений и ответов. Хотя для создания тела сообщения можно использовать любой текстовый редактор, система поможет в составлении адреса и многочисленных полей заголовков, добавляемых к каждому сообщению. Например, при составлении ответа на сообщение система электронной почты может извлечь адрес отправителя из полученного письма и автоматически поместить его в нужное место в ответе.

*Передача* — перемещение сообщений от отправителя к получателю. Для этого требуется установить соединение с адресатом или с какой-либо промежуточной машиной, переслать сообщение и разорвать соединение. Система электронной почты должна выполнять все эти действия автоматически, не беспокоя пользователя.

*Уведомление* — информирование отправителя о состоянии сообщения. Что с ним стало? Доставлено оно, потеряно или отвергнуто? Существует множество приложений, в которых подтверждение доставки имеет большую важность и даже может иметь юридическую значимость («Ваша честь, моя электронная система не очень надежна, поэтому я полагаю, что повестка с вызовом в суд просто где-то потерялась»).

*Отображение* приходящих сообщений на экране необходимо, чтобы пользователи имели возможность читать свою электронную почту. Иногда требуется преобразование текста или вызов специальной программы просмотра, например, для просмотра файла формата PostScript или прослушивания оцифрованного звукового сообщения. Иногда также применяются простые преобразования и форматирование текста.

*Обработка*. Наконец, на последнем этапе работы с электронным письмом решается дальнейшая судьба полученного сообщения. Получатель может удалить его, не читая, удалить после прочтения, сохранить и т.д. Также должна быть обеспечена возможность найти полученное ранее письмо и прочитать его еще раз, переслать его другому адресату или обрабатывать полученную почту другим способом.

Помимо этих пяти основных услуг большинство систем электронной почты предоставляют великое множество дополнительных функций. Перечислим кратко некоторые из них. Когда пользователи переезжают с места на место или отсутствуют в течение некоторого срока, им может понадобиться автоматическая переадресация их почты.

Большинство систем позволяют пользователям создавать почтовые ящики для хранения приходящей почты. Для работы с почтовыми ящиками нужны специальные команды, позволяющие создавать и удалять почтовые ящики, исследовать их содержимое, добавлять и удалять сообщения и т.д.

Менеджерам корпораций часто бывает нужно послать сообщение всем своим подчиненным, заказчикам или поставщикам. Для облегчения выполнения этой задачи применяются списки рассылки, представляющие собой список адресов электронной почты. При отправлении сообщения с помощью списка рассылки всем адресатам, числящимся в этом списке, посылаются идентичные копии сообщения.

Среди других полезных дополнительных функций можно перечислить следующие: рассылка копий писем «под копирку» (Carbon copy), рассылка копий без уведомления о других получателях (Blind carbon copy), письма с высоким приоритетом, секретная (то есть зашифрованная) почта, возможность доставки письма альтернативному получателю, если основной временно недоступен, а также возможность перепоручать обработку почты секретарям.

Сегодня электронная почта получила широкое применение в промышленности для обмена информацией внутри компании, что делает возможным совместную работу над сложными проектами далеко удаленных друг от друга (возможно, даже находящихся в различных временных зонах) сотрудников. Более того, общение по электронной почте устраняет концентрацию



внимания на различиях в положении, возрасте и поле, часто мешающих непредвзятому рассмотрению идей. Блестящая идея, посланная студентом-практикантом по электронной почте, имеет шанс выиграть у идеи не столь блестящей, но высказанной вице-президентом компании.

В основе всех современных систем электронной почты лежит ключевая идея о разграничении конверта и содержимого письма. Конверт заключает в себе сообщение. Он содержит всю информацию, необходимую для доставки сообщения — адрес получателя, приоритет, уровень секретности и т. п. Все эти сведения отделены от самого сообщения. Агенты передачи сообщений используют конверт для маршрутизации, аналогично тому, как это делает обычная почтовая служба.

Сообщение внутри конверта состоит из двух частей: заголовка и тела письма. Заголовок содержит управляющую информацию для пользовательских агентов. Тело письма целиком предназначается для человека-получателя. Примеры конвертов и сообщений показаны на рис. 5.1.

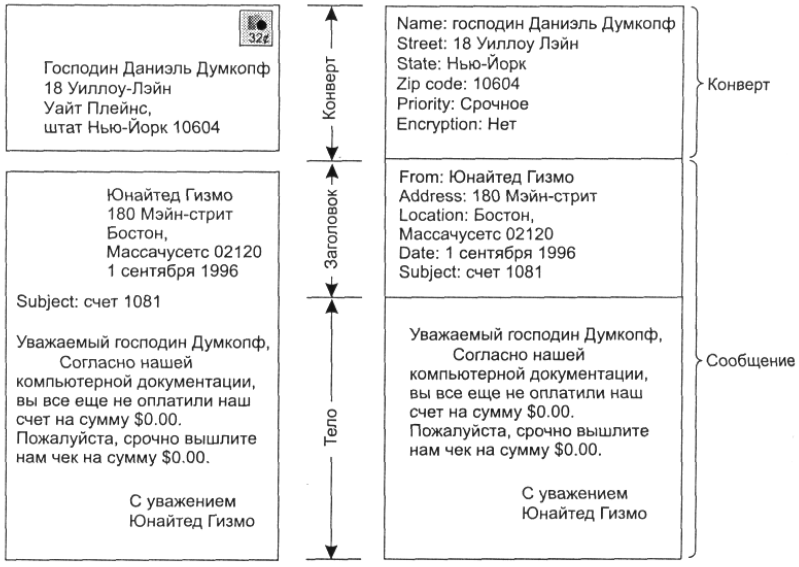



Рисунок 5.1 — Конверты и сообщения: обычное письмо (*а*); электронное письмо (*б*)

*Пользовательский агент*

Как уже было показано, системы электронной почты состоят из двух основных частей: пользовательских агентов и агентов передачи сообщений. В данном разделе мы рассмотрим пользовательских агентов. Как правило, пользовательский агент — это программа (иногда называемая почтовым редактором), управляемая множеством команд для составления и получения сообщений, а также для ответа на сообщения и управления почтовыми ящиками. Некоторые пользовательские агенты снабжены причудливым интерфейсом с использованием меню или графическим интерфейсом, для работы с которым требуется мышь, тогда как другие управляются односимвольными командами, вводимыми с клавиатуры. Функционально все они примерно одинаковы. В некоторых из них есть меню или ярлыки, но для основных действий обычно существуют комбинации клавиш.

*Отправление электронной почты*

Чтобы послать письмо электронной почтой, пользователь должен составить текст сообщения, указать адрес получателя, а также, возможно, некоторые дополнительные параметры. Текст сообщения может быть набран в отдельном текстовом редакторе, с помощью программы изготовления документов или в текстовом редакторе, встроенном в почтовый редактор. Адрес получателя должен быть указан в формате, понятном для пользовательского агента. Многие пользовательские агенты ожидают DNS-адреса вида полъзователъ@BM5-адрес.

Однако следует отметить, что существуют и другие формы адресации. В частности, адреса стандарта X.400 абсолютно не похожи на DNS-адреса и состоят из пар атрибут = значение, разделенных косыми чертами, например:

/C=US/ST=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR../CN=KEN SMITH/

В этом адресе указаны государство, штат, местоположение, личный адрес и имя получателя (Ken Smith). Возможно также использование различных других атрибутов, что делает возможным отправку электронного письма человеку, чьего имени вы не знаете, при условии, что вам известны другие атрибуты (например, название компании и должность получателя). Хотя



форма адресации X.400 значительно менее удобна, чем DNS, большинством систем электронной почты поддерживаются так называемые псевдонимы, с помощью которых пользователи могут вводить или выбирать имя адресата и получать корректный электронный адрес. Поэтому даже при использовании адресации X.400 пользователю не приходится вводить такие, мягко говоря, странные строки.

Большинством систем электронной почты поддерживаются списки рассылки, позволяющие пользователю рассылать одно и то же сообщение группе получателей при помощи одной команды. Если список рассылки хранится локально, пользовательский агент может просто послать каждому получателю отдельное сообщение. Однако при удаленном расположении списка рассылки сообщения будут тиражироваться там, где хранится список. Допустим, у группы исследователей птиц есть список рассылки, называющийся birders (птицеловы), установленный на домене meadowlark.arizona.edu. Тогда любое сообщение, посланное по адресу birders@meadowlark.arizona.edu, будет сначала отправляться в университет штата Аризона, а затем рассылаться оттуда в виде отдельных сообщений всем членам списка рассылки, где бы они ни находились. Для отправителя письма список рассылки внешне не отличается от обычного индивидуального адреса. Если отправитель не знает, что birders — это список рассылки, он вполне может подумать, что он посылает письмо лично некоему профессору по имени Gabriel O. Birders.

Каждая отображаемая строка содержит несколько полей, извлеченных из конверта или заголовка соответствующего сообщения. В простой системе электронной почты список отображаемых полей встроен в программу. В более сложных системах пользователь может выбрать отображаемые поля, а настройки пользователя будут храниться в специальном файле, называющемся профилем пользователя. В данном примере первое отображаемое поле — номер сообщения. Второе поле, Flags (флаги) может содержать флаг K, означающий, что сообщение не является новым, уже было прочитано и хранится в почтовом ящике; флаг A, означающий, что на данное сообщение уже был отправлен ответ; и/или флаг F, означающий, что сообщение было пере-

адресовано кому-то еще. Возможно также использование и других флагов.

Третье поле содержит размер сообщения в байтах, а в четвертом поле указывается отправитель сообщения. Поскольку значение этого поля просто извлекается из заголовка сообщения, это поле может содержать имена, полные имена, инициалы, имена регистрации в системе, а также все, что отправитель захочет указать в качестве своего имени. Наконец, поле Subject (тема) содержит краткое изложение содержания сообщения. Пользователи, забывающие заполнять поле Subject, часто обнаруживают, что их письма читаются респондентами далеко не в первую очередь.

*Чтение электронной почты*

Обычно при запуске пользовательский агент просматривает содержимое почтового ящика пользователя на предмет наличия новой почты. Затем он может объявить пользователю число новых сообщений в почтовом ящике или отобразить по одной строке сведений о каждом письме, после чего перейти в режим ожидания команды пользователя.

В качестве примера работы пользовательского агента рассмотрим типичный сценарий. Запустив пользовательский агент, пользователь запрашивает краткую сводку о своей почте. На экране при этом появляется список писем (таблица 5.1). Каждая строка соответствует одному полученному письму. В данном примере в почтовом ящике содержится восемь сообщений.

Таблица 5.1

| Флаги | Размер | Отправитель | Тема |
|---|---|---|---|
| K | 1030 | asw | Изменение в системе MINIX |
| KA | 6348 | vovka | Не все Вовки так уж противны |
| KF | 4519 | Amy N. Wong | Запрос сведений |
| | 1236 | bal | Биоинформатика |
| | 104110 | kaashoek | Материалы по одноранговым сетям |
| | 1223 | Frank | Re: Вы рассмотрите мой запрос на грант? |
| | 3110 | guido | Наша статья принята |
| | 1204 | dmr | Re: посещение моего студента |



После того как программа отобразила заголовки, пользователь может выполнить одну из нескольких команд: чтение, удаление письма и т.д. Старые системы с текстовым интерфейсом обычно управлялись с помощью односимвольных команд, таких как T (вывести сообщение), A (создать ответ), D (удалить сообщение) и F (переслать). Более современные системы имеют графический интерфейс. Обычно пользователь выбирает сообщение с помощью мыши, затем щелкает на значке, соответствующем определенному действию (выводу сообщения, созданию ответа, удалению и переадресации).

Электронная почта сильно изменилась с тех пор, когда она представляла собой простую передачу файлов. Пользовательские агенты с развитым набором услуг позволяют управляться с огромными потоками почты. Для всех, кому приходится получать и отправлять тысячи писем в год, такие программы просто незаменимы.

*Форматы сообщений*

Перейдем теперь от рассмотрения пользовательского интерфейса к формату самих сообщений электронной почты. Сначала мы рассмотрим основной ASCII-формат электронного письма стандарта RFC 822. Затем мы познакомимся с мультимедийным расширением этого стандарта.

*RFC 822*

Сообщения состоят из примитивного конверта (описанного в RFC 821), нескольких полей заголовка, пустой строки и, наконец, тела сообщения. Каждое поле заголовка (логически) состоит из одной строки ASCII-текста, содержащей имя поля, двоеточие и (в большинстве случаев) значение поля. RFC 822 был создан несколько десятилетий назад, и в нем нет четкого разграничения конверта и заголовка. Хотя частично стандарт был пересмотрен в RFC 2822, целиком обновить его было невозможно, поскольку RFC 822 уже был очень широко распространен. Обычно пользовательский агент формирует сообщение и передает его агенту передачи сообщений, который с помощью одного из полей заголовка создает конверт нового вида, представляющий собой некую старомодную смесь сообщения и конверта.

Основные поля заголовка, связанные с транспортировкой сообщения, перечислены в таблице 5.2 . Поле To: содержит DNS-адрес основного получателя. Возможно наличие и нескольких получателей. В поле Cc: указываются адреса дополнительных получателей. С точки зрения качества доставки, никакой разницы между основным и дополнительными получателями нет. Разница между ними чисто психологическая и может быть важна для людей, но совершенно не существует для почтовой системы. Термин Cc: (carbon copy — экземпляр, сделанный «под копирку») несколько устарел, так как при работе с компьютерами копировальная бумага вообще-то не используется, тем не менее, он прочно обосновался в электронной почте. Поле Bcc: (Blind carbon copy — слепая копия) аналогично предыдущему, с той разницей, что в последнем случае строка этого поля не видна получателям (как основному, так и дополнительным). Это свойство позволяет рассылать одно письмо одновременно нескольким получателям так, что получатели не будут знать, что это письмо послано еще кому-либо, кроме них.

Таблица 5.2 — Поля заголовка стандарта RFC 822, связанные с транспортировкой сообщения

| Поле | Значение |
|---|---|
| To: | Электронный адрес (адреса) основного получателя (получателей) |
| Cc: | Электронный адрес (адреса) дополнительного получателя (получателей) |
| Bcc: | Электронный адрес (адреса) слепой копии |
| From: | Автор (авторы) сообщения |
| Sender: | Электронный адрес отправителя |
| Received: | Строка, добавляемая каждым агентом передачи сообщений на протяжении маршрута |
| Return-Path: | Может быть использовано для идентификации обратного пути к отправителю |

Следующие два поля, From: и Sender:, сообщают, соответственно, кто составил и отправил сообщение. Это могут быть разные люди. Например, написать письмо может руководитель



предприятия, а отослать — его секретарша. В этом случае руководитель будет числиться в поле From:, а секретарша — в поле Sender. Поле From: является обязательным, тогда как поле Sender: может быть опущено, если его содержимое не отличается от содержимого поля From:. Эти поля нужны на случай, если сообщение доставить невозможно и об этом следует проинформировать отправителя. Кроме того, по адресам, указанным в этих полях, может быть оправлен ответ.

Строка, содержащая поле Received:, добавляется каждым агентом передачи сообщений на пути следования сообщения. В это поле помещаются идентификатор агента, дата и время получения сообщения, а также другая информация, которая может быть использована для поиска неисправностей в системе маршрутизации.

Поле Return-Path: добавляется последним агентом передачи сообщений. Предполагалось, что это поле будет сообщать, как добраться до отправителя. Теоретически эта информация может быть собрана из всех полей Received: заголовка (кроме имени почтового ящика отправителя), однако на практике оно редко заполняется подобным образом и обычно просто содержит адрес отправителя.

Помимо полей, показанных в таблице 5.2 , сообщения стандарта RFC 822 могут также содержать широкий спектр полей заголовка, используемых пользовательским агентом или самим пользователем. Наиболее часто используемые поля заголовка приведены в таблице 5.3 . Информации в таблице достаточно, чтобы понять назначение полей, поэтому мы не станем рассматривать их все подробно.

Таблица 5.3 — Некоторые поля, используемые в заголовке сообщения стандарта RFC 822

| Поле | Значение |
|---|---|
| Date: | Дата и время отправки сообщения |
| Reply-to: | Электронный адрес, на который следует присылать ответ |
| Message-Id: | Уникальный номер для последующей ссылки на это |

Окончание табл. 5.3

| Поле | Значение |
|---|---|
| | сообщение |
| In-Reply-To: | Идентификатор Message-Id сообщения, в ответ на которое посылается это сообщение |
| References: | Другие важные ссылки (идентификаторы Message-Id) |
| Keywords: | Ключевые слова, выбираемые пользователем |
| Subject: | Краткое изложение темы сообщения для отображения в одной строке |

Поле Reply-to: иногда используется в случае, если ни составитель письма, ни его отправитель не хотят получать на него ответ. Например, управляющий отделом сбыта пишет письмо, информирующее клиента о новом продукте. Это письмо отправляется его секретарем, но в поле Reply-to: указан адрес менеджера отдела продаж, который может ответить на вопросы и принять заказы.

В документе RFC 822 открыто сказано, что пользователям разрешается изобретать собственные заголовки для своих нужд при условии, что эти заголовки начинаются со строки X-. Гарантируется, что в будущем никакие стандартные заголовки не будут начинаться с этих символов, чтобы избежать конфликтов между официальными и частными заголовками. Иногда умники-студенты включают поля вроде X-Fruit-of-the-Day: (сегодняшний плод) или X-Disease-of-the-Week: (недуг недели), использование которых вполне законно, хотя их смысл и не всегда понятен.

После заголовков идет тело самого сообщения. Пользователь может разместить в нем все, что ему угодно. Некоторые люди завершают свои послания сложными подписями, включающими рисунки, созданные из ASCII-символов, популярными и малоизвестными цитатами, политическими заявлениями и разнообразными объявлениями (например, «Корпорация АБВ не несет ответственности за высказанное выше мнение. Собственно, она даже не в силах постичь его»).



*MIME — многоцелевые расширения электронной почты Интернет*

На заре существования сети ARPANET электронная почта состояла исключительно из текстовых сообщений, написанных на английском языке и представленных символами ASCII. Для подобного применения стандарта RFC 822 было вполне достаточно: он определял формат заголовков, но оставлял содержимое сообщения полностью на усмотрение пользователей. На сегодняшний день такой подход уже не удовлетворяет пользователей, привыкших к Интернету. Требуется обеспечить возможность отправления сообщений со следующими характеристиками.

1. Сообщения на языках с надстрочными знаками (например, на французском, немецком, испанском и т.д.).

2. Сообщения на языках, использующих алфавиты, отличные от латинского (например, на иврите или русском).

3. Сообщения на языках без алфавитов (например, китайском, японском, корейском).

4. Сообщения, вообще не являющиеся текстом (например, аудио или видео). Решение было предложено в документе RFC 1341, а более новая редакция была опубликована в RFC 2045-2049. Это решение, получившее название MIME (Multipurpose Internet Mail Extensions, — многоцелевые расширения электронной почты в Интернете), широко применяется в настоящий момент. Далее приведено его описание. Дополнительную информацию о наборе стандартов MIME смотрите в RFC.

Основная идея стандартов MIME — продолжить использование формата RFC 822, но с добавлением структуры к телу сообщения и с определением правил кодировки Не-ASCII-сообщений. Не отклоняясь от стандарта RFC 822, MIME-сообщения могут передаваться с помощью обычных почтовых программ и протоколов. Все, что нужно изменить, — это отправляющие и принимающие программы, которые пользователи могут создать для себя сами.

Стандартами MIME определяются пять новых заголовков сообщения, приведенных в таблице 5.4

Таблица 5.4 — Заголовки RFC 822, добавленные MIME

| Заголовок | Описание |
|---|---|
| MIME-Version: | Указывает версию стандартов MIME |
| Content-Description: | Описание содержимого. Строка обычного текста, информирующая о содержимом сообщения |
| Content-Id: | Уникальный идентификатор |
| Content-Transfer-Encoding: | Указывает способ кодировки тела сообщения для его передачи |
| Content-Type: | Тип и формат содержимого сообщения |

Первый заголовок просто информирует пользовательского агента, получающего сообщение, что тот имеет дело с сообщением MIME, а также сообщает ему номер версии MIME, используемой в этом сообщении. Если сообщение не содержит такого заголовка, то оно считается написанным на английском языке и обрабатывается соответствующим образом.

Заголовок Content-Description представляет собой ASCII-строку, информирующую о том, что находится в сообщении. Этот заголовок позволяет пользователю принять решение о том, нужно ли ему декодировать и читать сообщение. Если в строке сказано: «Фотография тушканчика Барбары», а получивший это сообщение не является любителем тушканчиков, то, вероятнее всего, он сразу удалит это сообщение, а не станет перекодировать его в цветную фотографию высокого разрешения.

Заголовок Content-Id содержит идентификатор содержимого сообщения. В нем используется тот же формат, что и в стандартном заголовке Message-Id:.

Заголовок Content-Transfer-Encoding сообщает о способе упаковки тела сообщения для его передачи по сети, которая может возражать против символов, отличных от букв, цифр и знаков препинания. Разработано пять схем (имеется возможность добавления новых схем).

Простейшая из них заключается в передаче просто ASCII-текста. Символы ASCII используют 7 разрядов и могут передаваться напрямую протоколом электронной почты при условии, что строка не превышает 1000 символов.



Следующая по простоте схема аналогична предыдущей, но использует 8-разрядные символы, то есть все значения байта от 0 до 255 включительно. Такая схема кодировки нарушает (исходный) протокол электронной почты Интернета, но используется в некоторых частях Интернета, в которых реализовано некоторое расширение исходного протокола. Хотя объявление о способе кодирования не делает его использование автоматически законным, открытое объявление может, по крайней мере, в случае чего объяснить неправильную работу почтовых агентов. Сообщения, использующие 8-разрядную кодировку, также должны соблюдать правило о максимальной длине строки.

Еще хуже обстоит дело с сообщениями в двоичной кодировке. К ним относятся произвольные двоичные файлы, которые не только используют все 8 разрядов в байте, но еще и не соблюдают ограничение на 1000 символов в строке. К этой категории относятся исполняемые программные файлы. Не дается никакой гарантии, что эти двоичные сообщения будут доставлены корректно, но, тем не менее, очень многие пользователи все равно пересылают их друг другу.

Корректный способ кодирования двоичных сообщений состоит в использовании кодировки base64 (64-символьная кодировка), иногда называемой ASCII armor (ASCII-оплетка). При использовании данного метода группы по 24 бита разбиваются на четыре 6-разрядные единицы, каждая из которых посылается в виде обычного разрешенного ASCII-символа. В этой кодировке 6-разрядный символ 0 кодируется ASCII-символом «A», 1 — ASCII-символом «B» и т.д. Затем следуют 26 строчных букв — это уже 10 разрядов, и наконец, + и / для кодирования 62 и 63 соответственно. Последовательности == и = говорят о том, что последняя группа содержит только 8 или 16 бит соответственно. Символы перевода строки и возврата каретки игнорируются, поэтому их можно вставлять в любом месте для того, чтобы строки выглядели не слишком длинными. Таким способом можно передать любой двоичный код.

Для сообщений, почти полностью состоящих из символов ASCII, но с небольшими включениями не-ASCII-символов, подобный метод несколько неэффективен. Вместо него лучше применять кодировку quoted-printable (цитируемое печатное ко-

дирование). Это просто 7-битный ASCII, в котором символы, соответствующие значениям ASCII-кода свыше 127, кодируются знаком равенства, за которым следуют две шестнадцатиричных цифры — ASCII-код символа.

Итак, двоичные данные следует посылать в кодировке Base64 или quoted-printable. Когда имеются веские причины не использовать эти методы, можно указать в заголовке Content-Transfer-Encoding: свою кодировку.

Последний заголовок в таблице 5.4 представляет наибольший интерес. Он указывает тип тела сообщения. В документе RFC 2045 определены семь типов содержимого сообщений, каждый из которых распадается на несколько подтипов. Подтип отделяется от типа косой чертой, например,

Content-Type: video/mpeg

Подтип должен быть явно указан в заголовке; подтипов по умолчанию нет. Начальный список типов и подтипов, определенный в документе RFC 2045, приведен в таблице 5.5 . С тех пор к ним было добавлено много новых типов и подтипов. Этот список пополняется всякий раз при возникновении соответствующей необходимости.

Таблица 5.5 — Типы стандарта MIME и подтипы, определенные в RFC 2045

| Тип | Подтип | Описание |
|---|---|---|
| Text | Plain | Неформатированный текст |
| | Enriched | Текст с включением простых команд форматирования |
| Image | Gif | Неподвижное изображение в формате GIF |
| | Jpeg | Неподвижное изображение в формате JPEG |
| Audio | Basic | Слышимый звук |
| Video | Mpeg | Видеофильм в формате MPEG |
| Application | Octet-stream | Неинтерпретируемая последовательность байтов |
| | Postscript | Документ для печати в формате PostScript |
| Message | Rfc822 | Сообщение MIME RFC 822 |
| | Partial | Сообщение разбито на части для передачи |
| | External-body | Само сообщение должно быть получено по |



Окончание табл. 5.5

| Тип | Подтип | Описание |
|---|---|---|
| | | сети |
| Multipart | Mixed | Независимые части в указанном порядке |
| | Alternative | То же сообщение в другом формате |
| | Parallel | Части сообщения следует просматривать одновременно |
| | Digest | Каждая часть является законченным сообщением стандарта RFC 822 |

Рассмотрим перечисленные в таблице типы сообщений. Тип text означает обычный текст. Комбинация text/plain служит для обозначения обычного текстового сообщения, которое может быть отображено на экране компьютера сразу после получения. Для этого не требуется дополнительной обработки или перекодировки. Это значение поля заголовка позволяет передавать обычные сообщения в MIME с добавлением небольшого количества дополнительных заголовков.

Подтип text/enriched позволяет включать в текст простой язык разметки документа. Этот язык обеспечивает системно-независимый способ выделять участки текста жирным или наклонным стилями, использовать шрифты самых разных размеров и цветов, отступы, выравнивание, верхние и нижние индексы и формировать простой макет страницы. В основе этого языка разметки лежит язык SGML (Standard Generalized Markup Language — стандартный обобщенный язык разметки), на базе которого был также создан язык HTML (HyperText Markup Language), применяемый в WWW.

Интерпретация зависит от принимающей сообщение системы. Если стили «жирный» и «курсив» доступны, они будут применены. Если нет, для выделения можно использовать подчеркивание, мигание, инверсную печать, выделение другим цветом и т.д. Разные системы могут применять и применяют свои стили.

Когда веб-технологии стали популярны, был добавлен (в RFC 2854) новый тип text/html, который позволил пересылать

веб-страницы в теле письма RFC 822. В RFC 3023 определен подтип для расширяемого языка разметки страниц, text/xml.

Следующим типом MIME является image. Он используется для передачи неподвижных изображений. На сегодняшний день существует множество различных форматов хранения и передачи изображений, как с использованием сжатия, так и без него. Два формата — GIF и JPEG — встроены практически во все браузеры, однако существует еще множество других, которые, надо полагать, вскоре дополнят этот список.

Типы audio и video предназначены, соответственно, для передачи звука и двигающегося изображения. Обратите внимание на то, что подтип video включает только визуальную информацию, а не звуковую дорожку. Если необходимо передать по электронной почте видеофильм со звуком, то, возможно, придется посылать видеоряд и звуковую дорожку отдельно друг от друга. Это зависит от системы кодирования. Первым видеоформатом, который был определен стандартом MIME, стал формат со скромным названием MPEG (Motion Pictures Experts Group — экспертная группа по вопросам движущегося изображения).

Тип application (приложение) предназначен для всех форматов, требующих внешней обработки, не обеспечиваемой ни одним из других типов. Тип octet-stream (байтовый поток) представляет собой просто последовательность никак не обрабатываемых байтов. Получив такой поток, пользовательский агент должен, вероятно, предложить пользователю сохранить его в виде файла и запросить для этого файла имя. Последующая обработка файла целиком зависит от пользователя.

Подтип postscript означает язык PostScript, созданный компанией Adobe Systems и широко используемый для описания страниц, предназначенных для печати. Многие принтеры имеют встроенные интерпретаторы языка PostScript. Хотя пользовательский агент может для отображения полученных PostScript-файлов просто вызвать внешний интерпретатор языка PostScript, подобные действия, вообще говоря, небезопасны. PostScript является полноценным языком программирования. При достаточном количестве свободного времени и некоторой склонности к самоистязанию на языке PostScript можно написать компилятор языка C или систему управления базами данных. Отображение



документа на языке PostScript осуществляется программой на языке PostScript, содержащейся в этом сообщении. Помимо отображения текста, эта программа способна читать, изменять или удалять файлы пользователя, а также может обладать еще целым рядом неприятных побочных эффектов.

Тип message позволяет помещать одно сообщение в другое. Это может быть полезно для переадресации письма. Если внутри одного сообщения заключено полное сообщение стандарта RFC 822, следует использовать подтип rfc822.

Подтип partial позволяет разбивать сообщение на отдельные части и передавать их отдельно (например, в случае, когда инкапсулированное сообщение является слишком длинным). Параметры обеспечивают восстановление сообщения получателем из отдельных фрагментов в правильном порядке.

Подтип external-body (внешнее тело) может применяться для очень длинных сообщений (таких, как видеофильмы). Вместо того чтобы помещать MPEG-файл в тело письма, в нем сообщается FTP-адрес, по которому пользовательский агент может получить этот файл тогда, когда нужно. Это особенно удобно в случае рассылки по списку рекламных роликов. При этом пользователям, не желающим просматривать ролик, не придется скачивать его по сети в свой почтовый ящик. Даже страшно подумать, что было бы, если бы по электронной почте стали рассылать спам в виде рекламных видеороликов.

Наконец, тип multipart позволяет составлять сообщение из нескольких частей, при этом начало и конец каждой части отчетливо указываются. Подтип mixed позволяет создавать сообщение из частей различных форматов. В случае применения подтипа alternative, напротив, каждая часть должна содержать одно и то же сообщение, но в другом виде или другой кодировке. Например, сообщение может быть послано в виде простого ASCII-текста, а также в форматах RTF и PostScript. Грамотно созданный пользовательский агент, получив такое сообщение, сначала попытается отобразить его в формате PostScript. Если это по какой-либо причине невозможно, тогда производится попытка отобразить часть в формате RTF. Если и это невозможно, отображается ASCII-текст. Части следует располагать в порядке увеличения сложности, чтобы даже старые (до MIME) пользова-

тельские агенты смогли отобразить сообщение хотя бы в виде простого ASCII-текста.

Подтип alternative также может использоваться для сообщений, посылаемых одновременно на разных языках.

Мультимедийный пример приведен в листинге 5.1. В данном случае поздравление с днем рождения посылается одновременно в виде текста и в виде песни. Если у получателя есть возможность воспроизвести звуковой файл birthday.snd, пользовательский агент скачает этот файл с указанного адреса и воспроизведет его. В противном случае на экране получателя в полной тишине отобразится текст сообщения. Части письма разделены двойными дефисами, за которыми следует (определяемая пользователем) строка, указанная как значение параметра boundary (граница).

Листинг 5.1 Сообщение, состоящее из RTF-текста и альтернативы в виде звукового файла

```
From: elinor@abc.com
To: Carolyn@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abc.com>
Content-Type: multipart/alternative: boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Земля обошла вокруг Солнца целое число раз
Это преамбула. Пользовательский агент игнорирует ее. Ку-ку.

qwertyuiopasdfghjklzxcvbnm
Content-Type: text/enriched
Happy birthday to you
Happy birthday to you
Happy birthday dear <bold> Carolyn
Happy birthday to you
--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body
access-type="anon-ftp":
sity="bicycle.abc.com":
directory="pub":
name="birthday.snd"
content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```

Обратите внимание: заголовок Content-Type трижды встречается в данном сообщении. На верхнем уровне он указывает, что сообщение состоит из нескольких частей. Для каждой части



он сообщает ее тип и подтип. Наконец, в теле второй части сообщения он указывает пользовательскому агенту тип внешнего файла. Чтобы подчеркнуть это различие, мы использовали в последнем случае строчные символы, хотя для всех заголовков регистр символа не имеет значения. Для внешнего тела в формате, отличном от 7-разрядного ASCII, также требуется заголовок content-transfer-encoding.

Для сообщений, состоящих из отдельных частей, существует еще два подтипа. Подтип parallel используется в том случае, когда требуется одновременный «просмотр» всех частей сообщения. Например, часто бывает так, что в фильмах звуковой канал отделен от изображения, однако эти два канала удобнее воспроизводить одновременно, а не последовательно.

Наконец, подтип digest используется, когда много сообщений упаковываются в одно сообщение. Например, какая-нибудь дискуссионная группа в Интернете может собирать сообщения от подписчиков, а затем высылать их в виде единого сообщения типа multipart/digest.

*Пересылка писем*

Система пересылки писем занимается доставкой электронных сообщений от отправителя получателю. Для этого проще всего установить транспортное соединение от машины-источника до машины-приемника, а затем просто передать по нему сообщение. Познакомимся с тем, как это осуществляется в действительности, и рассмотрим несколько ситуаций, в которых подобный подход не работает, и обсудим способы разрешения возникающих в связи с этим проблем.

*SMTP — простой протокол электронной почты*

В Интернете для доставки электронной почты машина-источник устанавливает TCP-соединение с портом 25 машины-приемника. Этот порт прослушивается почтовым демоном, и их общение происходит с помощью протокола SMTP (Simple Mail Transfer Protocol — простой протокол электронной почты). Этот демон принимает входящие соединения и копирует сообщения из них в соответствующие почтовые ящики. Если письмо невоз-

можно доставить, отправителю возвращается сообщение об ошибке, содержащее первую часть этого письма.

Протокол SMTP представляет собой простой ASCII-протокол. Установив TCP-соединение с портом 25, передающая машина, выступающая в роли клиента, ждет запроса принимающей машины, работающей в режиме сервера. Сервер начинает диалог с того, что посылает текстовую строку, содержащую его идентификатор и сообщающую о его готовности (или неготовности) к приему почты. Если сервер не готов, клиент разрывает соединение и повторяет попытку позднее.

Если сервер готов принимать почту, клиент объявляет, от кого поступила почта и кому она предназначается. Если получатель почты существует, сервер дает клиенту добро на пересылку сообщения. Затем клиент посылает сообщение, а сервер подтверждает его получение. Контрольные суммы не проверяются, так как транспортный протокол TCP обеспечивает надежный байтовый поток. Если у отправителя есть еще почта, она также отправляется. После передачи всей почты в обоих направлениях соединение разрывается. Пример диалога клиента и сервера при передаче сообщения из листинга 5.1 показан в листинге 5.2. Строки, посланные клиентом, помечены буквой C:, а посланные сервером — S:.

Листинг 5.2 Передача сообщения от elinor@abc.com для carolyn@xyz.com

```
S: 220 xyz.com.служба SMTP готова
C: Helo abc.com
S: 250 xyz.com приветствует abc.com
C: MAIL FROM: <elinor@abc.com>
S: 250 подтверждаю отправителя
C: RCPT TO: <Carolyn@xyz.com>
S: 250 подтверждаю получателя
C: DATA
S: 354 Отправляйте письмо: конец письма обозначается строкой, состоящей из символа “.”
C: From: elinor@abc.com
C: To: Carolyn@xyz.com
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@abc.com>
C: Content-Type: multipart/alternative: boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Земля обошла вокруг Солнца целое число раз
C:
C: Это преамбула. Пользовательский агент игнорирует ее. Ку-ку
```



```
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type:  text/richtext
C:
C: Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Carolyn </bold>
C: Happy birthday to you
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body:

C: access-type=”anon-ftp”:
C: site=”bicycle.abc.com”:
C: directory=”pub”:
C: name=:”birthday.snd”
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
S: 250 сообщение принято
C: QUIT
S: 221 xyz.com разрываю соединение
```

Несколько комментариев к листингу 5.2. Сначала клиент, естественно, посылает приветствие серверу. Таким образом, первая команда клиента выглядит как *HELO,* что представляет собой наиболее удачный из двух вариантов сокращения слова *HELLO* до четырех символов. Зачем все эти команды было нужно сокращать до четырех букв, сейчас уже никто не помнит.

В нашем примере сообщение должно быть послано только одному получателю, поэтому используется только одна команда *RCPT* (сокращение от слова recipient — получатель). Использование этой команды несколько раз позволяет посылать одно сообщение нескольким получателям. Каждое из них подтверждается или отвергается индивидуально. Несмотря на то, что попытки пересылки сообщения некоторым получателям оказываются неудачными (например, из-за отсутствия адресатов), это сообщение все равно может быть доставлено остальным адресатам, числящимся в списке рассылки.

Наконец, хотя синтаксис четырехсимвольных команд строго определен, синтаксис ответов не столь строг. Правила определяют только числовой код в начале строки. Все, что следует за

этим кодом, может считаться комментарием и зависит от конкретной реализации протокола.

Чтобы лучше понять, как работают SMTP и другие рассмотренные протоколы, попробуйте сами поработать с ними. В любом случае, для начала найдите машину, подключенную к Интернету. В системе UNIX наберите в командной строке:

**telnet mail.isp.com 25**

подставив вместо *mail.isp.com* DNS-имя почтового сервера провайдера. В системе Windows щелкните на кнопке Пуск, затем на кнопке Выполнить и наберите команду в диалоговом окне. В результате выполнения этой команды будет установлено telnet-соединение (то есть соединение TCP) с портом 25 данной машины. Как было показано в табл. 6.3, порт 25 является SMTP-портом. В ответ на введенную команду вы получите примерно следующее:

**Trying 192.30.200.66…**
**Connected to mail.isp.com**
**Escape character is '^]'.**
**220 mail.isp.com Small #74 ready at Thu. 25 Sept 2002 13:26 +0200**

Первые три строки посылаются telnet и поясняют для вас происходящее. Последняя строка посылается сервером SMTP удаленной машины и сообщает о готовности к общению с вашей машиной и приему почты. Чтобы узнать о доступных командах, наберите HELP.

Начиная с этого момента, возможен обмен последовательностями команд, показанными в листинге 5.2. Начинаться общение должно с команды клиента HELO.

Стоит отметить, что использование строк ASCII-текста в качестве команд не случайно. Большинство протоколов Интернета работают именно таким образом. Применение ASCII-текста упрощает тестирование и отладку протоколов. Тестирование можно производить, набирая команды вручную, как было показано ранее. Легко читаются выведенные в ответ сообщения.

Несмотря на то, что протокол SMTP определен довольно четко, все же могут возникать определенные проблемы. Одна из них связана с длиной сообщений. Некоторые старые реализации не поддерживали сообщения длиннее 64 Кбайт. Еще одна проблема связана с тайм-аутами. Если таймеры клиента и сервера настроены на разное время ожидания, один из них может внезапно разорвать соединение, в то время как противоположная сторона будет продолжать передачу. Наконец, иногда могут возникать бесконечные «почтовые штормы». Например, если хост 1 хранит список рассылки *A, a* хост 2 — список рассылки *B* и они содержат записи друг о друге, то письмо, посланное по одному из списков, будет создавать бесконечный объем трафика, пока кто-нибудь не заметит это.



Для решения некоторых из этих проблем был разработан расширенный протокол SMTP, ESMTP. Он описан в RFC 2821. Клиенты, желающие использовать его, должны начинать сессию связи с посылки приветствия EHLO вместо HELO. Если команда не принимается сервером, значит, сервер поддерживает только обычный протокол SMTP и клиенту следует работать в обычном режиме. Если же EHLO принято, значит, установлена сессия ESMTP и возможна работа с новыми параметрами и командами.

*Доставка сообщений*

До сих пор предполагалось, что все пользователи работают на машинах, способных посылать и получать электронную почту. Электронная почта доставляется, когда отправитель устанавливает TCP-соединение с получателем и посылает по нему сообщения. Такая модель прекрасно работала тогда, когда все хосты сети ARPANET (а позднее — Интернет) были, по сути, постоянно на линии и могли принимать TCP-соединения.

Однако с появлением пользователей, связывающихся с провайдерами с помощью модема, такой подход перестал оправдывать себя. Проблема вот в чем: что будет, если Элинор захочет отправить письмо Кэролайн, а та в данный момент не работает в Интернете? Получается, что Элинор не сможет установить TCP-соединение с Кэролайн, следовательно, невозможно будет запустить протокол SMTP, и Кэролайн так и не получит поздравление с днем рождения.

Одно из решений заключается в создании агента передачи сообщений на машине провайдера, который бы принимал и хра-

нил почту для своих пользователей в их почтовых ящиках. Поскольку такой агент может быть на линии постоянно, электронная почта может отправляться ему круглосуточно.

*POP3*

К сожалению, такое решение создает новую проблему: как пользователю забрать свою почту у агента передачи сообщений провайдера? Ответ таков: следует создать специальный протокол, который позволил бы пользовательскому агенту (на машине клиента) соединиться с агентом передачи сообщений провайдера (на машине провайдера) и скопировать хранящуюся для него почту. Одним из таких протоколов является POP3 (Post Office Protocol v. 3 — почтовый протокол, 3-я версия), определенный в документе RFC 1939.

При доставке сообщений возможны две ситуации:

первая — когда и отправитель и получатель постоянно подключены к Интернет (рис. 5.2,*а*);

вторая ситуация — когда отправитель подключен к линии, а получатель — нет (рис. 5.2,*б*).

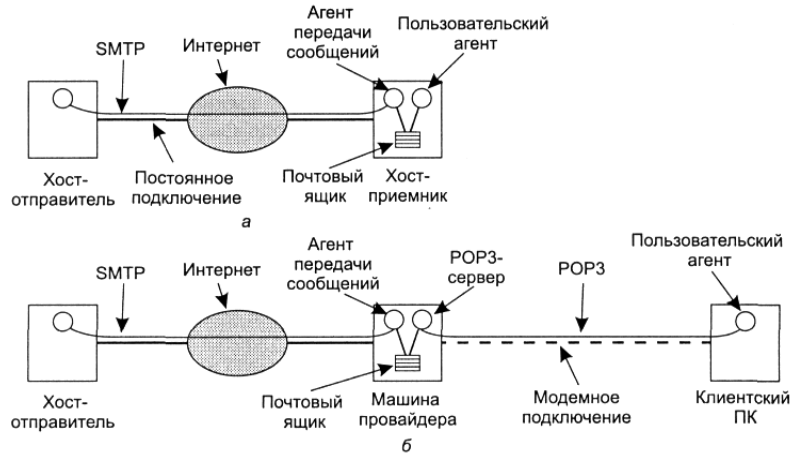



Рисунок 5.2 — Отправка и прием почты, когда приемник постоянно находится в подключенном состоянии и пользовательский агент работает на одной машине с агентом передачи сообщений (*а*); прием почты при модемном соединении получателя с провайдером (*б*)



Протокол POP3 запускается, вместе с почтовым клиентом. Последний дозванивается до провайдера (если только машина уже не находится в подключенном состоянии) и устанавливает TCP-соединение с агентом передачи сообщений с использованием порта 110. После установки соединения протокол POP3 проходит три последовательных состояния.

1. Авторизация.
2. Транзакции.
3. Обновление.

Авторизация связана с процессом входа пользователя в систему. В состоянии транзакций пользователь забирает свою почту и может пометить ее для удаления из почтового ящика. В состоянии обновления происходит удаление помеченной корреспонденции.

Можно посмотреть, как все это происходит, набрав команду вида

telnet mail.isp.com 110,

где *mail.isp.com* следует заменить на DNS-имя почтового сервера провайдера. Telnet устанавливает TCP-соединение с портом 110, прослушиваемым POP3-сервером. После установки TCP-соединения сервер посылает ASCII-сообщение, объявляя о своем присутствии. Обычно оно начинается с +OK, затем следует комментарий. Возможный сценарий после установки TCP-соединения показан в листинге 5.3. Как и раньше, строки, начинающиеся с C:, говорят о том, что данная команда исходит от клиента (пользователя), а начинающиеся с S: — что это сообщения сервера (агента передачи сообщений на машине провайдера).

Листинг 5.3 Получение трех сообщений по протоколу POP3

```
S: +OK POP3-сервер готов
C: USER Carolyn
S: OK
C: PASS vegetables
S: OK вход в систему
C: LIST
S: 1 2505
S: 2 14302
S: 3 8122
S: .
C: RETR 1
S: (отправляет сообщение 1)
```

```
C: DELE 1
C: RETR 2
S: (отправляет сообщение 2)
C: DELE 2
C: RETR 3
S: (отправляет сообщение 3)
C: DELE 3
C: QUIT
S: +OK Конец соединения с POP3-сервером
```

В состоянии авторизации клиент должен сообщить имя пользователя и пароль. После успешного входа в систему клиент может послать команду LIST для запроса списка писем, хранящихся в почтовом ящике. Каждая строка списка соответствует одному письму, в ней указываются его номер и размер. Точка является признаком конца списка.

После этого пользователь может запросить сообщения командой RETR и пометить их для удаления командой DELE. После получения (и, возможно, установки меток удаления) всех писем пользователь посылает команду QUIT для завершения состояния транзакций и входа в состояние обновления. После удаления сервером всех сообщений он посылает ответ и разрывает TCP-соединение.

Несмотря на то, что протокол POP3 действительно поддерживает возможность получения одного или нескольких писем и оставления их на сервере, большинство программ обработки электронной почты просто скачивают все письма и опустошают почтовый ящик на сервере. Такие действия означают, что реально хранится только одна копия писем — на жестком диске пользователя. Если с ним что-то случается, корреспонденция пропадает безвозвратно.

Теперь подведем небольшие итоги того, как происходит работа с электронной почтой клиентов провайдера. Элинор создает сообщение для Кэролайн с помощью редактора электронной почты (то есть пользовательского агента) и щелкает на значке, чтобы отослать его. Программа передает письмо агенту передачи сообщений на хосте Элинор. Агент передачи сообщений видит, что письмо адресовано carolyn@xyz.com, и использует DNS для поиска записи *MX* для *xyz.com* (где *xyz.com* — провайдер Кэролайн). В ответ на запрос возвращается DNS-имя почтового



сервера *xyz.com.* Агент передачи сообщений после этого снова обращается к DNS (например, используя gethostbyname): на этот раз ему нужно найти IP-адрес этой машины. Затем с помощью порта 25 найденной машины устанавливается TCP-соединение с SMTP-сервером. Передавая команды SMTP, аналогичные показанным в листинге 5.2, агент пересылает сообщение в почтовый ящик для Кэролайн и разрывает TCP-соединение.

Через некоторое время Кэролайн загружает свой компьютер, соединяется с провайдером и запускает программу электронной почты. Та устанавливает TCP-соединение через порт 110 POP3-сервера, работающего на машине провайдера. Имя DNS или IP-адрес этой машины обычно указывается при установке программы электронной почты, либо его получают у провайдера. После установки TCP-соединения почтовая программа Кэролайн запускает протокол POP3 для копирования содержимого почтового ящика на локальный жесткий диск. При этом происходит обмен командами, аналогичными командам в листинге 5.3. По окончании передачи электронной почты TCP-соединение разрывается. На самом деле в тот же момент можно разорвать и соединение с провайдером, поскольку вся почта уже находится на жестком диске у Кэролайн. Конечно, чтобы отправить ответ на письма, Кэролайн придется снова соединяться с провайдером, поэтому не всегда пользователи разрывают соединение сразу после загрузки почты.

*IMAP*

Пользователю, имеющему одну учетную запись у одного провайдера и всегда соединяющемуся с провайдером с одной и той же машины, вполне достаточно протокола POP3. Этот протокол и используется повсеместно благодаря его простоте и надежности. Однако в компьютерной индустрии есть такое незыблемое правило: если имеется нечто, что работает безупречно, всегда найдется некто, который захочет снабдить это нечто дополнительными возможностями (и тем самым снабдить его ошибками). Так произошло и с электронной почтой. У многих пользователей есть одна учетная запись в учебном заведении или на работе, но они хотят иметь доступ к ней и из дома, и с работы (учебы), и во время поездок (с портативного компьютера), и из интернет-кафе

во время так называемого отпуска. Хотя POP3 и предоставляет возможность разрешения такой ситуации (так как с его помощью все могут получить всю хранящуюся почту), но проблема в том, что корреспонденция пользователя очень быстро распространится более или менее случайным образом по всем машинам, с которых он получает доступ в Интернет, и некоторые из этих машин могут даже не принадлежать этому пользователю.

Это неудобство привело к созданию альтернативного протокола доставки сообщений, IMAP (Interactive Mail Access Protocol — протокол интерактивного доступа к электронной почте), определенного в RFC 2060. В отличие от протокола POP3, который подразумевает, что пользователь будет очищать почтовый ящик после каждого контакта с провайдером и будет работать с почтой в отключенном режиме, протокол IMAP предполагает, что вся почта будет оставаться в почтовых ящиках на сервере неограниченно долго. IMAP обладает широким набором механизмов для чтения сообщений или даже частей сообщений. Такое свойство полезно при использовании медленных модемов, поскольку можно прочесть только текстовую часть письма, к которому приложены большие видео- и аудиофрагменты. Поскольку основное предположение состоит в том, что пользователь не будет копировать на свой компьютер письма, в IMAP входят также инструменты для создания, удаления и других видов управления почтовыми ящиками, размещающимися на сервере. Таким образом, пользователь может завести собственный почтовый ящик для каждого лица, с которым ведется переписка, и переносить сообщения из почтового ящика для всех входящих писем в эти персональные ящики.

Протокол IMAP обладает разнообразными возможностями, например, способностью упорядочивать почту не по порядку ее поступления, как показано в таблице 5.1, а по атрибутам писем (например, «сначала дайте мне письмо от Бобби»). В отличие от POP3, IMAP может заниматься как доставкой исходящей почты от пользователя в направлении места назначения, так и доставлять входящую почту пользователя.

В целом стиль протокола IMAP подобен POP3. Различаются они количеством команд — в IMAP их десятки. Сервер IMAP прослушивает порт 143. Сравнение протоколов POP3 и IMAP



приведено в таблице 5.1. Следует заметить, что не все провайдеры и не все программы работы с электронной почтой поддерживают оба протокола. Поэтому, выбирая программу и провайдера, следует выяснить, могут ли они работать хоть с одним из этих протоколов, и если да, то с какими именно протоколами.

Таблица 5.1 — Сравнение протоколов POP3 и IMAP

| Свойство | POP3 | IMAP |
|---|---|---|
| Где определен | RFC 1939 | RFC 2060 |
| Используемый порт TCP | 110 | 143 |
| Место хранения почты | ПК пользователя | Сервер |
| Способ чтения почты | В автономном режиме | В подключенном режиме |
| Требуемое время нахождения на линии | Небольшое | Большое |
| Использование ресурсов сервера | Минимальное | Значительное |
| Поддержка нескольких почтовых ящиков | Отсутствует | Есть |
| Кто делает резервные копии почтовых ящиков | Пользователь | Провайдер |
| Удобство для мобильных пользователей | Нет | Да |
| Контроль загружаемой почты пользователем | Низкий | Полный |
| Возможность частичной загрузки сообщений | Нет | Есть |
| Наличие проблем с нехваткой места на диске | Нет | Есть (у провайдера) |
| Простота реализации | Да | Нет |
| Популярность | Широкая | Растет |

*Особенности доставки*

Независимо от того, куда доставляется почта — напрямую на рабочую станцию пользователя или на удаленный сервер, — многие системы предоставляют средства дополнительной обработки поступающей почты. Особенно большую ценность для пользователей представляет возможность устанавливать фильтры — наборы правил, выполняющихся при получении нового

письма или при запуске пользовательского агента. Каждое правило определяет некоторое условие и действие при его выполнении. Например, правило может гласить, что любое сообщение от начальника следует помещать в почтовый ящик номер 1, любое сообщение, пришедшее от кого-либо из группы друзей, следует помещать в почтовый ящик номер 2, а любое сообщение, содержащее определенные неприятные слова в поле *Subject,* следует удалять без вопросов.

Некоторые провайдеры предоставляют фильтры для борьбы со спамом. Эти фильтры автоматически классифицируют приходящие сообщения как нормальные или как спам (нежелательная реклама) и сохраняют каждое письмо в соответствующем ящике. Обычно такие фильтры определяют спам по адресу отправителя, если он является известным распространителем нежелательной почты. Затем анализируется поле *Subject.* Если сотни пользователей только что получили письмо с одинаковыми темами, возможно, это спам. Известны и другие методы выявления спама.

Еще одна предоставляемая услуга — возможность (временной) переадресации приходящей почты по другому адресу. Этим адресом может быть даже компьютер, управляемый коммерческой пейджинговой службой, передающей пользователю на пейджер, по радио или через спутник строку *Subject* каждого сообщения.

Еще одной часто используемой услугой доставки писем является возможность установки специального каникулярного демона. Это программа, отсылающая в ответ на приходящие письма сообщение типа Привет! Я в отпуске. Вернусь 24 августа. Желаю веселых каникул!

В подобных ответах можно также указывать способы решения срочных проблем, помещать адреса людей, которые могут решить специфические проблемы, и т. п. Большинство каникулярных демонов обычно формируют список лиц, которым они посылали подобные заранее заготовленные ответы, и воздерживаются от отправки такого ответа повторно тому же лицу. Грамотно написанные демоны также проверяют, не было ли полученное сообщение прислано по списку рассылки, и если да, то



вообще не отвечают на него. (Люди, посылающие сообщения в большие списки рассылки в летние месяцы, обычно не любят получать в ответ сотни сообщений с подробностями планов на отпуск каждого адресата.)

*Веб-почта*

Нельзя под конец не сказать нескольких слов о веб-почте. Некоторые веб-сайты, например Hotmail и Yahoo!, предоставляют электронную почту всем желающим. Работает эта система следующим образом. Имеются обычные агенты передачи сообщений, прослушивающие порт 25 в ожидании входящих SMTP-соединений. Чтобы связаться, например, со службой Hotmail, вам необходимо получить DNS-запись *MX.* Это можно сделать, например, набрав в командной строке UNIX

```
host -a -v Hotmail.com
```

Если предположить, что почтовый сервер называется *mx10.hotmail.com,* то для установки TCP-соединения, по которому можно привычным образом обмениваться командами SMTP, следует набрать:

```
telnet mx10.hotmail.com 25
```

Итак, ничего необычного в этой процедуре нет, если не принимать во внимание то, что большие серверы часто бывают заняты и иногда приходится предпринимать несколько попыток установки TCP-соединения.

Интересен здесь вопрос доставки электронной почты. Обычно, когда пользователь заходит на веб-страницу почтового сервера, он видит форму, в которой ему нужно заполнить поля *Имя пользователя* и *Пароль.* После того как он щелкает на кнопке *Войти,* пароль и имя пользователя отправляются на сервер, где проверяется их правильность. Если вход в систему осуществлен корректно, сервер находит почтовый ящик пользователя и строит список, подобный таблице 5.1, только оформленный в виде веб-страницы на HTML. Эта веб-страница отсылается на браузер клиента. Пользователь может щелкать на многих элементах страницы, выполняющих функции работы с почтовым ящиком, — чтения, удаления писем и т.д.

## 5.2 Всемирная паутина (WWW)

Всемирная паутина (WWW, World Wide Web) — это архитектура, являющаяся основой для доступа к связанным между собой документам, находящимся на миллионах машин по всему Интернету. За 10 лет своего существования из средства распространения информации на тему физики высоких энергий она превратилась в приложение, о котором миллионы людей с разными интересами думают, что это и есть «Интернет». Огромная популярность этого приложения стала следствием цветного графического интерфейса, благодаря которому даже новички не встречают затруднений при его использовании. Кроме того, Всемирная паутина предоставляет огромное количество информации практически по любому вопросу, от африканских муравьедов до яшмового фарфора.

Всемирная паутина была создана в 1989 году в Европейском центре ядерных исследований CERN (Conseil Europeen pour la Recherche Nucleaire) в Швейцарии. В этом центре есть несколько ускорителей, на которых большие группы ученых из разных европейских стран занимаются исследованиями в области физики элементарных частиц. В эти команды исследователей часто входят ученые из пяти-шести и более стран. Эксперименты очень сложны, для их планирования и создания оборудования требуется несколько лет. Программа Web (паутина) появилась в результате необходимости обеспечить совместную работу находящихся в разных странах больших групп ученых, которым нужно было пользоваться постоянно меняющимися отчетами о работе, чертежами, рисунками, фотографиями и другими документами.

Изначальное предложение, создать паутину из связанных друг с другом документов пришло от физика центра CERN Тима Бернерс-Ли (Tim Berners-Lee) в марте 1989 года. Первый (текстовый) прототип заработал спустя 18 месяцев. В декабре 1991 году на конференции Hypertext'91 в Сан-Антонио в штате Техас была произведена публичная демонстрация.

Эта демонстрация, сопровождаемая широкой рекламой, привлекла внимание других ученых. Марк Андрессен (Marc Andreessen) в университете Иллинойса начал разработку первого графического браузера, Mosaic. Программа увидела свет в феврале 1993 года и стала настолько популярной, что год спустя ее автор Марк Андрессен решил сформировать собственную компанию Netscape Communications Corp., чьей целью была разработка клиентов, серверов и другого программного обеспечения для веб-приложений. Когда в 1995 году корпорация Netscape получила известность, инвесторы, полагая, очевидно, что появилась еще одна корпорация типа Microsoft, заплатили за пакет ее акций 1,5 млрд. долларов. Это было тем более удивительно, что номенклатура продукции компании состояла всего из одного наименования, компания имела устойчивый пассивный баланс, а в своих проспектах корпорация Netscape объявила, что в обозримом будущем не рассчитывает на получение прибыли. В течение последующих трех лет между Netscape Navigator и Internet Explorer от Microsoft развернулась настоящая «война браузеров». Разработчики с той и с другой стороны в безумном порыве пытались напичкать свои программы как можно большим числом функций (а следовательно, и ошибок) и в этом превзойти соперника. В 1998 году America Online купила корпорацию Netscape Communications за 4,2 млрд долларов, положив таким образом конец весьма непродолжительному независимому существованию компании.



В 1994 году CERN и Массачусетский технологический институт (M.I.T., Massachusetts Institute of Technologies) подписали соглашение об основании WWW-консорциума (World Wide Web Consortium, иногда применяется сокращение W3C) — организации, цель которой заключалась в дальнейшем развитии приложения Web, стандартизации протоколов и поощрении взаимодействия между отдельными сайтами. Бернерс-Ли стал директором консорциума. Хотя о Всемирной паутине уже написано очень много книг, лучшее место, где вы можете получить самую свежую информацию о ней, это сама Всемирная паутина. Домашнюю страницу консорциума можно найти по адресу http://www.w3.org. На этой странице заинтересованный читатель найдет ссылки на другие страницы, содержащие информацию обо всех документах консорциума и о его деятельности.

### 5.2.1 Архитектура WWW

С точки зрения пользователя Всемирная паутина состоит из огромного собрания документов, расположенных по всему миру. Документы обычно называют для краткости просто страницами. Каждая страница может содержать ссылки (указатели) на другие связанные с ней страницы в любой точке мира. Пользователи могут следовать по ссылке (например, просто щелкнув на ней мышью), при этом страница, на которую указывает ссылка, загружается и появляется в окне браузера. Этот процесс можно повторять бесконечно. Идея страниц, связанных между собой гиперссылками (гипертекст), была впервые пророчески предложена в 1945 году, задолго до появления Интернета, Ванневаром Бушем (Vannevar Bush), профессором из Массачусетского университета, занимавшимся электротехникой.

Страницы просматриваются специальной программой, называемой браузером. Самыми популярными браузерами являются программы Internet Explorer и Netscape. Браузер предоставляет пользователю запрашиваемую страницу, интерпретирует ее текст и содержащиеся в нем команды форматирования для вывода страницы на экран. Большинство веб-страниц начинается с заголовка, содержит некоторую информацию и заканчивается адресом электронной почты организации, отвечающей за состояние этой страницы. Строки текста, представляющие собой ссылки на другие страницы, называются гиперссылками. Они обычно выделяются либо подчеркиванием, либо другим цветом, либо и тем и другим. Чтобы перейти по ссылке, пользователь перемещает указатель мыши в выделенную область, при этом меняется форма указателя, и щелкает на ней. Хотя существуют и текстовые браузеры, как, например, Lynx, они не так популярны, как графические, поэтому мы сконцентрируем наше внимание на последних. Следует заметить, что также разрабатываются браузеры, управляемые голосом.

Рассмотрим основные принципы работы Паутины (рис. 5.3). Браузер отображает веб-страницу на клиентской машине. Когда пользователь щелкает на строке, которая является ссылкой на страницу, расположенную на сервере abcd.com, браузер следует по этой гиперссылке. Реально при этом на abcd.com отправляется служебное сообщение с запросом страницы. Получив страницу, браузер показывает ее. Если на этой странице содержится гиперссылка на страницу с сервера xyz.com, то браузер обращается с запросом к xyz.com, и так далее до бесконечности.



Давайте теперь более детально рассмотрим сторону клиента, основываясь на рис. 5.3. По сути дела, браузер — это программа, которая может отображать вебстраницы и распознавать щелчки мыши на элементах активной страницы.

При выборе элемента браузер следует по гиперссылке и получает с сервера запрашиваемую страницу. Следовательно, гиперссылка внутри документа должна быть устроена так, чтобы она могла указывать на любую страницу Всемирной паутины. Страницы именуются с помощью URL (Uniform Resource Locator — унифицированный указатель информационного ресурса). Типичный указатель выглядит так:

http://www.abcd.com/products.html.

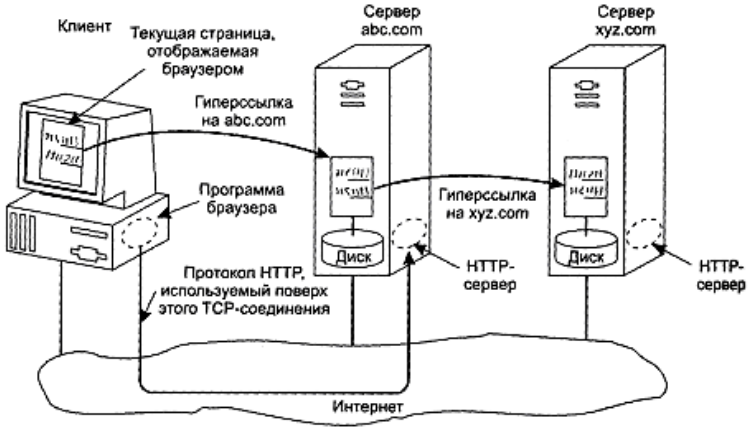



Рисунок 5.3 — Части Всемирной паутины

*Сторона клиента*

Более подробно про унифицированные указатели URL будет рассказано далее в этой главе. Пока что достаточно знать, что URL состоит из трех частей: имени протокола (http), DNS-имени машины, на которой расположена страница (www.abcd.com), и (обычно) имени файла, содержащего эту страницу

(products.html). Между щелчком пользователя и отображением страницы происходят следующие события.

Когда пользователь щелкает мышью на гиперссылке, браузером выполняется ряд действий, приводящих к загрузке страницы, на которую указывает ссылка. Предположим, пользователь, блуждая по Интернету, находит ссылку на документ, рассказывающий про интернет-телефонию, а конкретно — на домашнюю страницу ITU, расположенную по адресу http://www.itu.org/home/index.html. Рассмотрим каждое действие, происходящее после выбора этой ссылки.

1. Браузер определяет URL (по выбранному элементу страницы).
2. Браузер запрашивает у службы DNS IP-адрес www.itu.org.
3. DNS дает ответ 156.106.192.32.
4. Браузер устанавливает TCP-соединение с 80-м портом машины 156.106.192.32.
5. Браузер отправляет запрос на получение файла /home/index..html.
6. Сервер www.itu.org высылает файл /home/index.html.
7. TCP-соединение разрывается.
8. Браузер отображает весь текст файла /home/index.html.
9. Браузер получает и выводит все изображения, прикрепленные к данному файлу'.
10. Многие браузеры отображают текущее выполняемое ими действие в строке состояния внизу экрана. Это позволяет пользователю понять причину низкой производительности: например, не отвечает служба DNS или сервер или просто сильно перегружена сеть при передаче страницы.
11. Для отображения каждой новой страницы браузер должен понять ее формат. Чтобы все браузеры могли отображать любые страницы, они пишутся на стандартизованном языке HTML, описывающем веб-страницы. Более детально мы рассмотрим его далее.

Несмотря на то, что браузер, по сути дела, представляет собой интерпретатор HTML, большинство браузеров оснащаются многочисленными кнопками и функциями, облегчающими навигацию по Всемирной паутине. У многих браузеров есть кнопки для возврата на предыдущую страницу и перехода на следующую страницу (последняя доступна только в том случае, если пользователь уже возвращался назад), а также кнопка для прямого перехода на домашнюю страницу пользователя. Большинство браузеров поддерживают в меню команды для установки закладки на текущей странице и отображения списка закладок, что позволяет попадать на любую страницу при помощи всего одного щелчка мышью. Страницы также могут быть сохранены на диске или распечатаны на принтере. Кроме того, доступны многочисленные функции управления отображением страницы и установки различных настроек пользователя.



Помимо обычного текста (не подчеркнутого) и гипертекста (подчеркнутого), веб-страницы могут также содержать значки, рисунки, чертежи и фотографии. Все они могут быть связаны ссылкой с другой страницей. Щелчок на элементе, содержащем ссылку, также вызовет смену текущей страницы, отображаемой браузером. С большими изображениями, такими как фотографии или карты, может быть ассоциировано несколько ссылок, при этом следующая отображаемая страница будет зависеть от того, на каком месте изображения произведен щелчок мышью.

Далеко не все страницы написаны исключительно с применением HTML. Например, страница может состоять из документа в формате PDF, значка в формате GIF, фотографии в JPEG, звукозаписи в формате MP3, видео в MPEG или чего-то еще в одном из сотни форматов. Поскольку стандартные HTML-страницы могут иметь ссылки на любые файлы, у браузера возникает проблема обработки страницы, которую он не может интерпретировать.

Вместо того чтобы наращивать возможности и размеры браузеров, встраивая в них интерпретаторы для различных типов файлов (количество которых быстро растет), обычно применяется более общее решение. Когда сервер возвращает в ответ на запрос какую-либо страницу, вместе с ней высылается некоторая дополнительная информация о ней. Эта информация включает MIME-тип страницы (таблица 5.5 ). Страницы типа text/html выводятся браузером напрямую, как и страницы некоторых других встроенных типов. Если же для данного MIME-типа внутренняя интерпретация невозможна, браузер определя-

ет, как выводить страницу, по своей таблице MIME-типов. В данной таблице в соответствии с каждым типом ставится программа просмотра.

Существуют два способа отображения: с помощью подключаемого модуля (plug-in) или вспомогательных приложений. Подключаемый модуль представляет собой особый код, который браузер извлекает из специального каталога на жестком диске и устанавливает в качестве своего расширения (рис. 5.4,*а*). Поскольку подключаемые модули работают внутри браузера, у них есть доступ к текущей странице, вид которой они могут изменять. После завершения своей работы, обычно это связано с переходом пользователя на другую страницу, подключаемый модуль удаляется из памяти браузера.

Каждый браузер имеет набор процедур, которые должны реализовывать все подключаемые модули. Это нужно для того, чтобы браузер мог обращаться к последним. Например, существует стандартная процедура, с помощью которой базовый код браузера передает подключаемому модулю данные для отображения. Набор этих процедур образует интерфейс подключаемого модуля и является специфичным для каждого конкретного браузера.

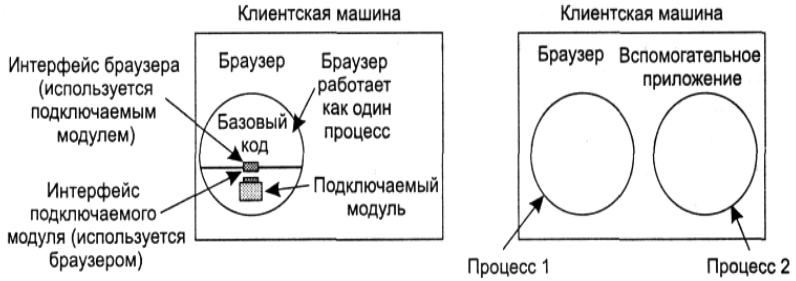



Рисунок 5.4 — Браузер с подключаемым модулем (а); вспомогательное приложение (б)

Кроме того, браузер предоставляет подключаемому модулю определенный набор своих процедур. Среди них в интерфейс браузера обычно включаются процедуры распределения и освобождения памяти, вывода сообщений в строке статуса браузера и опроса параметров браузера.



Перед использованием подключаемого модуля его нужно установить. Этот процесс подразумевает, что пользователь копирует файл установки с веб-сайта производителя модуля. В Windows он обычно представляет собой самораспаковывающийся архив ZIP с расширением .exe. Если дважды щелкнуть на таком файле, запускается небольшая программа, включенная в начало архива. Она распаковывает подключаемый модуль и копирует его в соответствующий каталог, известный браузеру. Затем она регистрирует MIME-тип, обрабатываемый модулем, и ассоциирует этот тип с модулем. В системах UNIX установочный файл зачастую представляет собой основной сценарий, осуществляющий копирование и регистрацию.

Вторым способом расширения возможностей браузера является использование вспомогательных приложений. Вспомогательное приложение — это полноценная программа, работающая как отдельный процесс (рис. 5.4,*б*). Поскольку она никак не связана с браузером, между ними отсутствует какой бы то ни было интерфейс. Вместо этого обычно вспомогательное приложение вызывается с параметром, представляющим собой имя временного файла, содержащего данные для отображения. Получается, что браузер можно настроить на обработку практически любого типа файлов, не внося в него никаких изменений. Современные веб-серверы часто содержат сотни комбинаций типов и подтипов файлов. Новые типы файлов появляются всякий раз при установке новых программ.

Вспомогательные приложения не обязательно связаны только с MIME-типом application (приложение). Например, Adobe Photoshop будет работать с image/x-photoshop, а RealOne Player может поддерживать audio/mp3.

В Windows каждая устанавливаемая на компьютер программа регистрирует типы, которые она хочет поддерживать. Такой механизм приводит к конфликту, когда несколько программ могут обрабатывать один и тот же подтип, например, video/mpg. Конфликт разрешается следующим образом: программа, которая регистрируется последней, затирает своей записью существующую ассоциацию (тип MIME, вспомогательное приложение) для тех типов, которые она готова обрабатывать самостоятельно. Следствием этого является то, что каждая уста-

навливаемая программа может изменить метод отображения браузером некоторых типов.

В UNIX этот процесс обычно не является автоматическим. Пользователь должен вручную обновлять конфигурационные файлы. Такой подход приводит к уменьшению числа неожиданных сюрпризов, но при этом у пользователя появляются дополнительные заботы.

Браузеры могут работать и с локальными файлами, не запрашивая информацию с удаленных серверов. Поскольку локальные файлы не сообщают свои MIME-типы, браузеру нужно каким-то хитрым образом определить, какой подключаемый модуль или вспомогательное приложение использовать для типов, отличных от встроенных (таких, как text/html или image/jpeg). Для поддержки локальных файлов вспомогательные модули и приложения должны быть ассоциированы как с расширениями файлов, так и с типами MIME. При стандартных настройках попытка открыть файл foo.pdf приведет к его открытию в Acrobat, а файл bar.doc будет открыт в Word. Некоторые браузеры для определения типа MIME используют не только данные о типе MIME, но и расширение файла и даже данные, взятые из самого файла. В частности, Internet Explorer по возможности старается ориентироваться на расширение файла, а не на тип MIME.

Здесь также возможны конфликты, поскольку многие программы страстно желают поддерживать, например, .mpg. Профессиональные программы при установке обычно выводят флажки, позволяющие выбрать поддерживаемые типы MIME и расширения. Таким образом, пользователь может выбрать то, что ему требуется, и таким образом избежать случайного затирания существующих ассоциаций. Программы, нацеленные на массового потребителя, полагают, что большинство пользователей понятия не имеют о типах MIME и просто захватывают все что могут, совершенно не обращая внимания на ранее установленные программы.

Расширение возможностей браузера по поддержке новых типов файлов — это удобно, но может также привести к возникновению некоторых проблем. Когда Internet Explorer получает файл с расширением exe, он думает, что это исполняемая программа и никаких вспомогательных средств для нее не требует-



ся. Очевидно, следует просто запустить программу. Однако такой подход может оказаться серьезной дырой в системе защиты информации. Злоумышленнику требуется лишь создать нехитрый сайт с фотографиями, скажем, знаменитых киноактеров или спортсменов и поставить ссылки на вирусы. Один-единственный щелчок мышкой на фотографии может в этом случае привести к запуску непредсказуемой и, возможно, опасной программы, которая будет действовать на машине пользователя. Для предотвращения подобных нежелательных ситуаций Internet Explorer можно настроить на избирательный запуск неизвестных программ, однако не все пользователи в состоянии справиться с настройкой браузера.

В UNIX похожая проблема может возникнуть с основными сценариями, однако при этом требуется, чтобы пользователь сознательно стал устанавливать вспомогательную программу. К счастью, это процесс довольно сложный, и «случайно» установить что-либо практически невозможно (немногие могут сделать это и преднамеренно).

*Сторона сервера*

О стороне клиента сказано уже достаточно много. Поговорим теперь о стороне сервера. Как мы уже знаем, когда пользователь вводит URL или щелкает на гиперссылке, браузер производит структурный анализ URL и интерпретирует часть, заключенную между http:// и следующей косой чертой, как имя DNS, которое следует искать. Вооружившись IP-адресом сервера, браузер устанавливает TCP-соединение с портом 80 этого сервера. После этого отсылается команда, содержащая оставшуюся часть URL, в которой указывается имя файла на сервере. Сервер возвращает браузеру запрашиваемый файл для отображения.

В первом приближении веб-сервер напоминает сервер, представленный в листинге 4.2. Этому серверу, как и настоящему веб-серверу, передается имя файла, который следует найти и отправить. В обоих случаях в основном цикле сервер выполняет следующие действия:

1. Принимает входящее TCP-соединение от клиента (браузера).
2. Получает имя запрашиваемого файла.

3. Получает файл (с диска).
4. Возвращает файл клиенту.
5. Разрывает TCP-соединение.

Современные веб-серверы обладают более широкими возможностями, однако существенными в их работе являются именно перечисленные шаги.

Проблема данного подхода заключается в том, что каждый запрос требует обращения к диску для получения файла. В результате число обращений к веб-серверу за секунду ограничено максимальной скоростью обращений к диску. Среднее время доступа к высокоскоростному диску типа SCSI составляет около 5 мс, то есть сервер может обрабатывать не более 200 обращений в секунду. Это число даже меньше, если часто запрашиваются большие файлы. Для крупных веб-сайтов это слишком мало.

Очевидным способом решения проблемы является кэширование в памяти последних запрошенных файлов. Прежде чем обратиться за файлом к диску, сервер проверяет содержимое кэша. Если файл обнаруживается в кэше, его можно сразу выдать клиенту, не обращаясь к диску. Несмотря на то, что для эффективного кэширования требуются большие объемы памяти и некоторое дополнительное время на проверку кэша и управление его содержимым, суммарный выигрыш во времени почти всегда оправдывает эти накладные расходы и стоимость.

Следующим шагом, направленным на повышение производительности, является создание многопоточных серверов. Одна из реализаций подразумевает, что сервер состоит из входного модуля, принимающего все входящие запросы, и k обрабатывающих модулей (рис. 5.5). Все k + 1 потоков принадлежат одному и тому же процессу, поэтому у обрабатывающих модулей есть доступ к кэшу в адресном пространстве процесса. Когда приходит запрос, входящий модуль принимает его и создает краткую запись с его описанием. Затем запись передается одному из обрабатывающих модулей. Другая возможная реализация подразумевает отсутствие входного модуля; все обрабатывающие модули пытаются получить запросы, однако здесь требуется блокирующий протокол, помогающий избежать конфликтов.



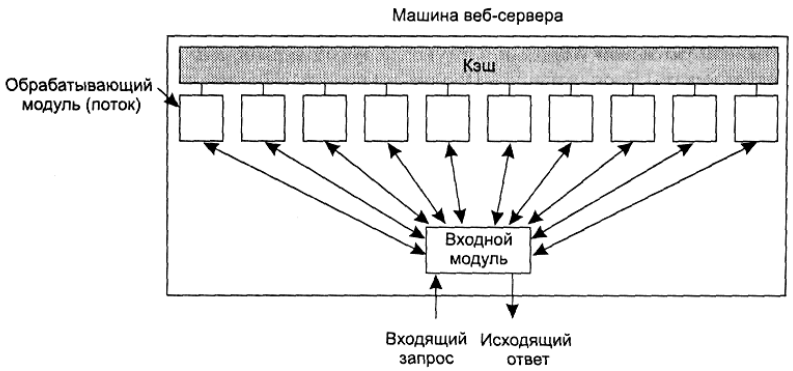



Рисунок 5.5 — Многопоточный веб-сервер с входным и обрабатывающими модулями

Обрабатывающий модуль вначале проверяет кэш на предмет нахождения там нужных файлов. Если они там действительно есть, он обновляет запись, включая в нее указатель на файл. Если искомого файла в кэше нет, обрабатывающий модуль обращается к диску и считывает файл в кэш (при этом, возможно, затирая некоторые хранящиеся там файлы, чтобы освободить место). Считанный с диска файл попадает в кэш и отсылается клиенту.

Преимущество такой схемы заключается в том, что пока один или несколько обрабатывающих модулей заблокированы в ожидании окончания дисковой операции (при этом такие модули не потребляют мощности центрального процессора), другие модули могут активно обрабатывать захваченные ими запросы. Разумеется, реального повышения производительности за счет многопоточной схемы можно достичь, только если установить несколько дисков, чтобы в каждый момент времени можно было обращаться более чем к одному диску. Имея k обрабатывающих модулей и k дисков, производительность можно повысить в k раз по сравнению с однопоточным сервером и одним диском.

Теоретически, однопоточный сервер с k дисками тоже должен давать прирост производительности в k раз, однако программирование и администрирование такой схемы оказывается

очень сложным, так как в этом случае невозможно использование обычных блокирующих системных вызовов READ для чтения с диска. Многопоточные серверы такого ограничения не имеют, поскольку READ будет блокировать только тот поток, который осуществил системный вызов, а не весь процесс.

Современные веб-серверы выполняют гораздо больше функций, чем просто прием имен файлов и отправка файлов. На самом деле, реальная обработка каждого запроса может оказаться довольно сложной. По этой причине на многих серверах каждый обрабатывающий модуль выполняет серии действий. Входной модуль передает каждый входящий запрос первому доступному модулю, который обрабатывает его путем выполнения некоторого подмножества указанных далее шагов в зависимости от того, что именно требуется для данного запроса:

- вычисление имени запрашиваемой веб-страницы;
- регистрация клиента;
- осуществление контроля доступа для клиента;
- осуществление контроля доступа для веб-страницы;
- проверка кэша;
- получение запрошенной страницы с диска;
- определение типа MIME для включения этой информации в ответ клиенту;
- аккуратное выполнение различных дополнительных задач;
- возвращение ответа клиенту;
- добавление записи в журнал активности сервера.

Шаг 1 необходим, потому что входящий запрос может и не содержать реального имени файла в виде строкового литерала. Например, URL может быть вот таким: http://www.cs.vu.nl. Здесь имя файла отсутствует. Этот URL необходимо дополнить неким именем файла по умолчанию. К тому же современные браузеры могут указывать язык пользователя по умолчанию (например, итальянский или английский), что позволяет серверу выбирать веб-страницу на соответствующем языке, если таковая существует. Вообще говоря, расширение имени — задача не такая уж тривиальная, как может показаться, поскольку существует множество соглашений об именовании файлов.

Шаг 2 состоит в проверке идентификационных данных клиента. Это нужно для отображения страниц, недоступных для



широкой публики. Мы обсудим один из способов такой проверки далее в этой главе.

Шаг 3 проверяет наличие каких-либо ограничений, накладываемых на данного клиента и его местоположение. На шаге 4 проверяются ограничения на доступ к запрашиваемой странице. Если определенный файл (например, .htaccess) присутствует в том же каталоге, что и нужная страница, он может ограничивать доступ к файлу. Например, можно установить доступ к странице только для сотрудников компании.

Шаги 5 и 6 включают в себя получение страницы. Во время выполнения шага 6 должна быть обеспечена возможность одновременного чтения с нескольких дисков.

Шаг 7 связан с определением типа MIME, исходя из расширения файла, первых нескольких байтов, конфигурационного файла или каких-то иных источников. Шаг 8 предназначен для различных задач, таких как построение профиля пользователя, сбор статистики и т.д.

На шаге 9 наконец отсылается результат, что фиксируется в журнале активности сервера на шаге 10. Последний шаг требуется для нужд администрирования. Из подобных журналов можно впоследствии узнать ценную информацию о поведении пользователей — например, о том, в каком порядке люди посещают страницы на сайте.

Если приходит слишком много запросов в секунду, центральный процессор может перестать справляться с их обработкой вне зависимости от того, сколько дисков параллельно работают на сервере. Решается эта проблема установкой на сервере нескольких узлов (компьютеров). Их полезно укомплектовывать реплицированными (содержащими одинаковую информацию) дисками во избежание ситуации, когда узким местом становится дисковый накопитель. В результате возникает многомашинная система, организованная в виде серверной фермы (рис. 5.6). Входной модуль по-прежнему принимает входящие запросы, однако распределяет их на сей раз не между потоками, а между центральными процессорами, снижая тем самым нагрузку на каждый компьютер. Отдельные машины сами по себе могут быть многопотоковыми с конвейеризацией, как и в рассматриваемом ранее случае.

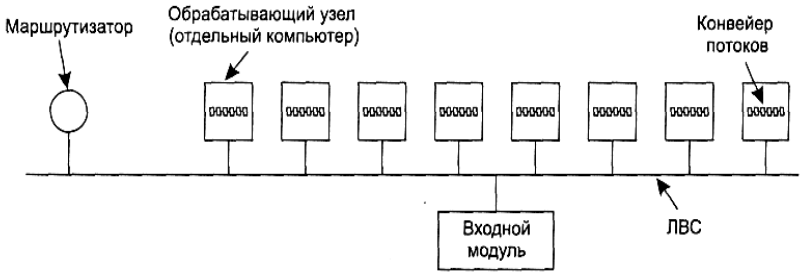



Рисунок 5.6 — Серверная ферма

Одна из проблем, связанных с серверными фермами, заключается в отсутствии общего кэша — каждый обрабатывающий узел обладает собственной памятью. Эта проблема может быть решена установкой дорогостоящей мультипроцессорной системы с разделяемой памятью, однако существует и более дешевый способ. Он заключается в том, что входной модуль запоминает, на какой узел он посылал запросы конкретных страниц. Последующие запросы тех же страниц он сможет тогда направлять на те же узлы. Таким образом, получается, что каждый узел специализируется по своему набору страниц; и отпадает необходимость хранения одних и тех же файлов в кэшах разных компьютеров.

Другая проблема, возникающая при использовании серверных ферм, состоит в том, что TCP-соединение клиента заканчивается на входном модуле, то есть ответ в любом случае должен пройти через входной модуль (рис. 5.7,*а*). Здесь как входящий запрос (1), так и исходящий ответ (2) проходят через входной модуль. Иногда для обхода этой проблемы применяется хитрость под названием передача TCP. Суть ее в том, что TCP-соединение продлевается до конечного (обрабатывающего) узла, и он может самостоятельно отправить ответ напрямую клиенту (рис. 5.7,*б*). Эта передача соединения для клиента незаметна.



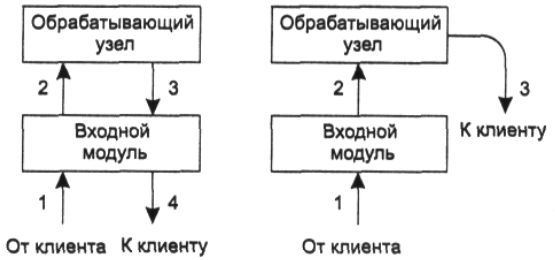



Рисунок 5.7 — Обычный запрос-ответный обмен (а); обмен запросами и ответами при передаче TCP (б)

*URL — унифицированные указатели информационных ресурсов*

Мы несколько раз упоминали о том, что веб-страницы могут быть связаны между собой ссылками. Пора познакомиться с тем, как эти ссылки реализованы. Уже при создании Паутины было очевидно, что для реализации ссылок с одних страниц на другие необходим механизм именования и указания расположения страниц. В частности, прежде чем выводить выбранную страницу на экран, нужно узнать ответы на три следующих вопроса.

1. Как называется эта страница?
2. Где она расположена?
3. Как получить к ней доступ?

Если бы каждой странице можно было присвоить уникальное имя, то в идентификации страниц не было бы никакой неоднозначности. Тем не менее, проблему бы это не решило. Для примера проведем параллель между страницами и людьми. В Соединенных Штатах почти у всех граждан есть номер карточки социального страхования, представляющий собой уникальный идентификатор, так как нет двух людей с одинаковым номером. Тем не менее, зная только номер карточки социального страхования, нет способа узнать адрес владельца и, конечно, нельзя определить, следует ли писать этому гражданину по-

английски, по-испански или по-китайски. Во Всемирной паутине проблемы, в принципе, те же самые.

В результате было принято решение идентифицировать страницы способом, решающим сразу все три проблемы. Каждой странице назначается унифицированный указатель информационного ресурса (URL, Uniform Resource Locator), который служит уникальным именем страницы. URL состоят из трех частей: протокола (также называемого схемой), DNS-имени машины, на которой расположена страница, и локального имени, единственным образом идентифицирующего страницу в пределах этой машины (обычно это просто имя файла).

Этот URL состоит из трех частей: протокола (http), DNS-имени хоста (www.cs.vu.nl) и имени файла (video/index-en.html). Отдельные части URL-указателя разделяются специальными знаками пунктуации. Имя файла представляет собой относительный путь по отношению к веб-катологу cs.vu.vu.nl.

У сайтов могут быть сокращенные имена для ускоренного доступа к определенным файлам. Скажем, при отсутствии в URL имени файла может выводиться главная (домашняя) страница сайта. Если имя файла заканчивается именем каталога, то из него по умолчанию выбирается файл с именем index.html. Наконец, имя — user/ может соответствовать WWW-каталогу пользователя, причем может быть также задано имя файла по умолчанию, например, index.html.

Теперь надо понять, как работает гипертекст. Чтобы на неком участке текста браузер мог реагировать на щелчок мыши, при написании веб-страницы нужно обозначить два элемента: отображаемый на экране текст ссылки и URL страницы, которая должна стать текущей при щелчке мышью. Синтаксис такой команды будет пояснен далее в этой главе.

При выборе ссылки браузер с помощью службы DNS ищет имя хоста. Зная IP-адрес хоста, браузер устанавливает с ним TCP-соединение. По этому соединению с помощью указанного протокола браузер посылает имя файла, содержащего страницу. Вот, собственно, и все. Назад по соединению передается страница.



Такая схема является открытой в том смысле, что она позволяет использовать разные протоколы для доставки информационных единиц разного типа. Определены URL-указатели для других распространенных протоколов, понимаемые многими браузерами. Слегка упрощенные формы наиболее употребительных URL-указателей приведены в таблице 5.7 .

Таблица 5.7 — Некоторые распространенные URL-указатели

| Имя | Применение | Пример |
|---|---|---|
| http | Гипертекст (HTML) | http://www.cs.vu.nl/~ast/ |
| ftp | FTP | ftp://ftp.cs.vu.nl/pub/minix/README |
| file | Локальный файл | file:////usr/suzanne/prog.c |
| news | Телеконференция | news:comp.os.minix |
| news | Статья новостей | news:AA01 342231 12@cs.utah.edu |
| gopher | Gopher | Gopher://gopher.tc.umn.edu/11/Libraries |
| mailto | Отправка электронной почты | mailto:JohnUser@acm.org |
| telnet | Удаленный терминал | telnet://www.w3. org:80 |

Кратко рассмотрим этот список. Протокол http является родным языком Всемирной паутины, на нем разговаривают веб-серверы. HTTP — это сокращение, которое расшифровывается как HyperText Transfer Protocol (протокол передачи гипертекста).

Протокол ftp используется для доступа к файлам по FTP — протоколу передачи файлов по Интернету. За двадцать лет своего существования он достаточно хорошо укоренился в сети. Многочисленные FTP-серверы по всему миру позволяют пользователям в любых концах Интернета регистрироваться на сервере и скачивать разнообразные файлы, размещенные на сервере. Всемирная паутина здесь не вносит особых изменений. Она просто упрощает доступ к FTP-серверам и работу с файлами, ибо само по себе FTP имеет несколько загадочный интерфейс (однако более мощный, чем HTTP: например, он позволяет пользователю машины A передать файл с машины B на машину C).

К локальному файлу также можно обратиться как к веб-странице, либо используя протокол file, либо просто указав имя файла. Такой подход напоминает FTP, но не требует наличия сервера. Разумеется, он работает только с локальными файлами, а не с расположенными на удаленных терминалах.

Задолго до появления Интернета появилась система групп новостей USENET. Она состоит примерно из 30 000 конференций, в которых миллионы людей обсуждают широкий круг вопросов, отправляя и читая сообщения, связанные с тематикой данной конференции. Протокол news позволяет пользователю вызывать на экран статью с новостями, как если бы она была обычной веб-страницей. Это означает, что веб-браузер легким движением руки превращается в элегантную программу чтения новостей. На самом деле, благодаря кнопкам и пунктам меню многих браузеров чтение новостей USENET становится даже удобнее, чем с помощью специальных программ чтения сетевых новостей.

Для протокола news поддерживается два формата URL-указателей. Первый формат указывает телеконференцию, и с его помощью можно получить список новых статей с указанного заранее сайта новостей. Второй формат позволяет получить конкретную статью по ее идентификатору, например, AA0134223112©cs. utah.edu. Для получения этой статьи с заранее настроенного сайта браузер использует протокол NNTP (Network News Transfer Protocol — сетевой протокол передачи новостей). Мы изучим NNTP в этой книге, однако надо понимать, что это нечто вроде SMTP, они весьма похожи даже по стилю.

Протокол gopher используется системой Gopher, разработанной в университете штата Миннесота и получившей свое название от университетской спортивной команды «Golden Gophers» («Золотые суслики»). (Гоферами называют уроженцев штатов Миннесота, Арканзас и Флорида. Кроме того, на американском сленге это слово означает «добывать», «копать», «искать».) Система Gopher появилась в Интернете на несколько лет раньше Всемирной паутины. Концептуально они похожи: и та, и



другая представляют собой схему поиска и получения информации, хранящейся на различных серверах, однако система Gopher поддерживала только тексты и не поддерживала изображений. Сейчас ее можно считать полностью устаревшей, используется она крайне редко.

Последние два протокола не занимаются имитацией получения веб-страниц, но они также полезны. Протокол mailto позволяет пользователю посылать электронную почту из веб-браузера. Например, в некоторых браузерах для этого нужно щелкнуть на кнопке OPEN и ввести URL-указатель, состоящий из слова mailto:, за которым следует почтовый адрес получателя. В ответ в большинстве браузеров откроется специальная форма, содержащая поля для редактирования темы письма и других заголовков, а также окно для ввода текста самого письма.

С помощью протокола telnet можно установить в подключенном режиме соединение с удаленным компьютером. Он используется так же, как и программа Telnet, что неудивительно, так как большинство браузеров просто вызывают саму программу Telnet как вспомогательное приложение.

Итак, URL-указатели позволяют пользователям не только путешествовать по Всемирной паутине, но и работать с FTP-серверами, BBS, Gopher-серверами, электронной почтой и регистрироваться на удаленных серверах с помощью программы Telnet. Все эти ресурсы оказываются доступны при помощи всего одной программы — веб-браузера, что очень удобно. Если бы отцом этой идеи не был ученый-физик, она стала бы, вероятно, самой убедительной рекламой какой-нибудь компании, специализирующейся на выпуске программного обеспечения.

Несмотря на все перечисленные достоинства, все продолжающийся рост популярности Всемирной паутины выявил один врожденный недостаток URL-схемы. URL указывает на определенный хост. Часто запрашиваемые по сети страницы было бы лучше дублировать и хранить копии в удаленных концах сети, чтобы снизить сетевой трафик. Беда в том, что URL-указатели не предоставляют возможности для ссылки на страницу без указания ее точного адреса. Нельзя сказать: «Мне нужна страница

abc, и мне все равно, где вы ее раздобудете». Для решения задачи репликации страниц проблемная группа проектирования Интернета IETF (Internet Engineering Task Force) работает над системой URN (Uniform Resource Name — универсальное имя ресурса). Универсальное имя ресурса URN можно считать обобщенным URL-указателем. В настоящее время этот вопрос находится в стадии исследования, хотя уже предложен синтаксис, описанный в RFC 2141.

### 5.2.2 Статические веб-документы

Основная идея Всемирной паутины состоит в перемещении веб-страниц от сервера клиенту. Простейшие веб-страницы являются статическими, то есть это просто файлы, размещенные на каком-либо сервере и ожидающие востребования. В этом контексте даже видео может быть статической страницей, поскольку это всего лишь файл. В этом разделе мы подробно рассмотрим статические вебстраницы. В следующем разделе нам предстоит изучение динамического наполнения страниц.

*HTML — язык разметки веб-страниц*

Веб-страницы на сегодняшний день пишутся на языке HTML (HyperText Markup Language). С помощью HTML можно размещать на веб-страницах текст, графику, а также указатели на другие страницы. Он является языком разметки, то есть языком, описывающим способ форматирования документа. Термин «разметка» (markup) восходит к тем дням, когда технический редактор с помощью специальной разметки указывал специалисту-типографу, какой шрифт использовать для печати документа. Таким образом, языки разметки содержат подробные команды форматирования. Например, в языке HTML, команда  означает начало участка текста, печатаемого полужирным шрифтом, а  означает конец такого участка. Преимущество языка разметки перед языком, не имеющим явных команд форматирования, заключается в том, что браузеры для отображения страниц, написанных на этом языке, программируются довольно



просто: браузер должен понимать и выполнять содержащиеся в тексте команды разметки. Среди других популярных примеров языков разметки — языки TeX и troff.

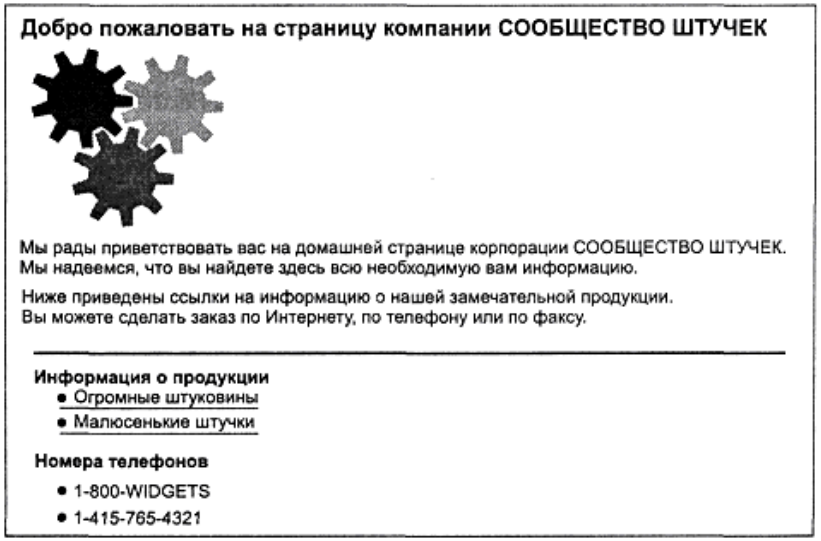
С помощью встроенных стандартизированных команд разметки в HTML-файлах становится возможным читать и переформатировать любую веб-страницу веб-браузером. Способность изменять форматирование важно, так как должна быть возможность просматривать веб-страницу, созданную на экране с установленным разрешением 1600x1200 точек при 24 битах на точку, на экране с разрешением, например, 640x320 точек при 8 битах на точку.

Далее мы приведем краткий обзор языка HTML, просто чтобы дать о нем представление. Хотя, в принципе, можно создавать HTML-документы с помощью стандартных текстовых редакторов, и многие так и делают, также есть возможность использовать специальные HTML-редакторы, берущие на себя большую часть работы (за счет снижения возможностей пользователя детально контролировать получаемый результат).

Веб-страница состоит из заголовка и тела. Вся страница размещается между командами форматирования, называемыми в языке HTML тегами, <html> и </html>. Впрочем, большинство браузеров правильно отобразят страницу и в отсутствие этих тегов. Заголовок веб-страницы заключен в скобки тегов <head> и </head>, а тело располагается между тегами <body> и </body> (рис. 5.8,*а*). Команды внутри тегов называют директивами. Большинство HTML-тегов имеют такой формат, то есть <something> помечает начало чего-либо, а </something> — его конец. Большинство браузеров предоставляют возможность просмотра исходного HTML-кода веб-страниц (пункт меню View Source или нечто подобное).

```
<HTML> <HEAD> <TITLE> Корпорация СООБЩЕСТВО ШТУЧЕК. </TITLE> </HEAD>
<BODY> <H1> Добро пожаловать на страницу компании СООБЩЕСТВО ШТУЧЕК. </H1>
<IMG SRC="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <BR>
Мы рады приветствовать вас на домашней странице корпорации <B> СООБЩЕСТВО ШТУЧЕК</B>
Мы надеемся, что <I> вы </I> найдете здесь всю необходимую вам информацию.
<P>Ниже приведены ссылки на информацию о нашей замечательной продукции.
Вы можете сделать заказ по Интернету, по телефону или по факсу. <HR>
<H2> Информация о продукции </H2>
<UL> <LI> <A HREF="http://widget.com/products/big"> Большущие штуковины </A>
<LI> <A HREF="http://widget.com/products/little"> Малюсенькие штучки </A>
</UL>
<H2> Номера телефонов </H2>
<UL> <LI> телефон: 1-800-WIDGETS
<LI> факс: 1-415-765-4321
</UL> </BODY> </HTML>
```

(а)




(б)

Рисунок 5.8 — Пример веб-страницы на HTML (а); форматированная страница (б)

Регистр символов в тегах не имеет значения. Например, <head> и <HEAD> означают одно и то же, однако новый стандарт требует использования исключительно строчных букв. Формат самого HTML-текста, то есть расположение строк и т.д., не имеет значения. Программы обработки HTML-текстов игнорируют лишние пробелы и переносы строк, так как они все равно форматируют текст так, чтобы он помещался в заданной области отображения. Соответственно для того чтобы исходные HTML-документы легче читались, в них можно добавлять произвольное количество знаков табуляции, пробелов и символов переноса строк. И наоборот, для разделения абзацев в тексте в исходный HTML-текст недостаточно вставить пустую строку, так как она просто игнорируется браузером. В этом случае необходимо явное использование специального тега.



Некоторые теги могут иметь (именованные) параметры, называемые атрибутами. Например: <img src=«abc» alt=«foobar»> представляет собой тег <img> с атрибутом src, которому присвоено значение «abc», и атрибутом alt, которому присвоено значение «foobar». Для каждого тега стандарт HTML устанавливает список допустимых атрибутов и их значение. Поскольку все атрибуты являются именованными, их порядок не имеет значения.

Формально при написании HTML-документов должен использоваться набор символов Latin-1 международного стандарта ISO 8859-1, но для пользователей, чьи клавиатуры поддерживают только ASCII-символы, для ввода специальных символов, таких как, например, «ё», могут использоваться специальные управляющие последовательности символов. Эти последовательности должны начинаться со знака амперсанда и заканчиваться точкой с запятой. Например, è означает символ é, а é — символ è. Так как сами символы <, > и & оказываются зарезервированными, для их отображения в тексте также применяются управляющие последовательности: < (less than — знак «меньше»), > (greater than — знак «больше») и &amp: (ampersand — амперсанд).

Главным пунктом заголовка является название страницы, располагающееся между тегами <title> и </title>. В него можно поместить также некоторую мета-информацию. Она не отображается на странице, а используется некоторыми браузерами для того, чтобы помечать окно страницы.

Рассмотрим некоторые другие команды языка HTML, используемые в примере на рис. 5.8, и другие, приведенные в таблице 5.8 . Заголовки в примере задаются тегами вида <hn>, где n — цифра от 1 до 6. <hl> является самым важным заголовком,

головком, <h6> — наименее важным. Как это отобразить на экране, зависит от браузера. Обычно заголовки с меньшими номерами отображаются более крупными шрифтами. Браузер может также выделять различные заголовки различными цветами. Обычно заголовки <hl> выводятся на экран крупным полужирным шрифтом и выделяются, по меньшей мере, одной пустой строкой над и под заголовком.

Таблица 5.8 — Наиболее часто используемые HTML-теги. У некоторых из них могут быть дополнительные параметры

| Тег | Описание |
|---|---|
| <html>...</html> | Объявляет веб-страницу на языке HTML |
| <head> ... </head> | Определяет границы заголовка страницы |
| <title>...</title> | Определяет границы неотображаемого названия страницы |
| <body> ... </body> | Определяет границы тела страницы |
| <hn> ... </hn> | Определяет границы заголовка уровня n |
|  ...  | Маркирует блок текста, печатаемого полужирным шрифтом |
| ...  | Маркирует блок текста, печатаемого курсивом |
|  ...  | Помечает начало и конец центрированного по горизонтали текста |
| <ul>...</ul> | Помечает начало и конец неупорядоченного списка |
| <ol> ...</ol> | Помечает начало и конец упорядоченного списка |
| <menu> ... </menu> | Помечает границы меню |
| <li> ... </li> | Маркирует начало и конец пункта меню |
| <br> | Разрыв (перевод строки) |
| <P> | Начало абзаца |
| <hr> | Горизонтальная линейка |
| <img src="..."> | Загрузка изображения |
| <a href="..."> ... </a> | Определяет гиперссылку |

Теги  и  обозначают, соответственно, полужирный шрифт (boldface) и курсив (italics). Если браузер не может отобразить полужирный шрифт или курсив, он должен применить



какой-нибудь другой способ выделения, например, использовать другой цвет или инверсное отображение символов.

Язык HTML предоставляет несколько механизмов создания списков, включая вложенные списки. Тег <ul> (unordered list) начинает неупорядоченный список. Отдельные пункты, помеченные в исходном тексте тегом <li>, изображаются с маркером абзаца (обычно крупной черной точкой, •) перед собой. Тег <ol> (ordered list) означает начало упорядоченного списка. При его использовании абзацы, помеченные тегом <li>, автоматически нумеруются браузером. У списков, организованных при помощи тегов <ul> и <o!>, одинаковый синтаксис (за исключением открывающих и закрывающих тегов списков) и похожее поведение.

Теги <br>,  и <hr> применяются для обозначения границ между различными участками текста. Точный формат может быть определен в таблице стилей (см. ниже), ассоциированной со страницей. Тег <br> просто вставляет разрыв строки. Обычно браузеры не вставляют пустую строку после тега <br>. Тег , напротив, начинает новый абзац, перед которым может быть вставлена пустая строка и, возможно, добавлен отступ. (Тег , отмечающий конец абзаца, существует, но на практике почти не используется. Большинство составителей HTML-страниц даже не знают о его существовании.) Наконец, тег <hr> прерывает строку и рисует на экране горизонтальную линию.

Язык HTML позволяет включать в веб-страницу изображения. Тег <img> указывает, что в данной позиции страницы должно быть загружено изображение. У этого тега может быть несколько параметров. Параметр src задает URL изображения. Стандартом HTML не определяются графические форматы. На практике все браузеры поддерживают файлы форматов GIF и JPEG. Браузеры могут поддерживать любые другие форматы, но эта свобода оказывается палкой о двух концах. Если пользователь привыкнет к браузеру, поддерживающему, скажем, формат файлов BMP, он может включить их в свои веб-страницы, а затем обнаружить, что остальные браузеры просто игнорируют всю его замечательную работу.

У тега <img> может быть еще несколько параметров. Параметр al ign управляет выравниванием изображения относительно текста. Он может принимать значения top (верх), middle

(центр), bottom (низ). Параметр alt предоставляет текст, отображаемый вместо изображения, если пользователь запретил вывод изображений. Параметр ismap является флагом, указывающим, является ли данное изображение активной картой.

И наконец, мы подошли к гиперссылкам, использующим пару тегов <a> (anchor — якорь) и </a>. У этого тега также могут быть различные параметры, из которых следует отметить href (гиперссылка, URL) и name (имя гиперссылки). Текст, располагающийся между тегами <a> и </a>, отображается на экране. Если этот текст выбирается, браузер открывает страницу, на которую указывает гиперссылка. Между тегами <a> и </a> можно также размещать изображение (тег <img>). В этом случае, если пользователь щелкнет на изображении, будет произведен переход по ссылке.

Для примера рассмотрим следующий фрагмент HTML-текста:

<a href=http://www.nasa.gov> Домашняя страница NASA </a>

При отображении страницы с этим фрагментом на экране появляется следующая строка:

Домашняя страница NASA

Если затем пользователь щелкнет мышью на этом тексте, браузер обратится по сети к указанному URL (http://www.nasa.gov) и попытается получить там вебстраницу и отобразить ее на экране.

В качестве второго примера рассмотрим следующую строку:

<a href=http://www.nasa.gov> <img src=«shuttle.gif» alt=«NASA»> </A>

При отображении этой страницы должно быть видно изображение (челночный воздушно-космический аппарат). Щелчок мышью на этом изображении будет иметь тот же результат, что и щелчок на подчеркнутом тексте в предыдущем примере. Если пользователь запретит автоматическое отображение изображений, вместо него будет показан текст «NASA».

Тег <a> может содержать параметр name, что позволяет создать гиперссылку посреди текста, на которую можно ссылаться из другого места этой же страницы. Например, некоторые веб-страницы начинаются с оглавления, состоящего из «локальных» гиперссылок. Щелчок мышью на пункте оглавления позволяет быстро переместиться в нужное место страницы.

HTML продолжает развиваться. В первых двух версиях не существовало таблиц, они были добавлены только в HTML 3.0.



HTML-таблица состоит из нескольких строк, каждая из которых состоит из нескольких ячеек. Ячейка может содержать широкий спектр данных, включая текст, изображения и даже другие таблицы. Ячейки могут объединяться вместе, например, заголовок таблицы может охватывать несколько столбцов. Контроль составителей страниц над внешним видом таблиц ограничен. Последнее слово в таких вопросах, как выравнивание, стили рамок и границы ячеек, остается за браузерами.

Реализация таблицы на языке HTML показана в листинге 5.4, а ее возможное отображение браузером — таблица 5.2. Этот пример демонстрирует несколько основных возможностей HTML-таблиц. Таблицы начинаются с тега <table>. Для описания основных свойств таблицы может быть предоставлена дополнительная информация.

Листинг 5.4 HTML-таблица

```
<html>
<head> <title> Пример страницы с таблицей </title> </head>
<body>
<table border=all rules=all>
<caption> Некоторые различия html версий </caption>
<col align=left>
<col align=center>
<col align=center>
<col align=center>
<tr> <th> Аспект <th> HTML 1.0 <th>HTML 2.0 <th> HTML 3.0 <th>HTML 4.0</tr>
<tr> <th> Гиперссылки <td> x <td> x<td> x </tr>
<tr> <th> Изображения <td> x <td> x<td> x </tr>
<tr> <th> Списки <td> x <td> x<td> x </tr>
<tr> <th> Активные карты и изображения <td> :<td>x<td> x<td> x</tr>
<tr> <th> Формы <td> :<td> x <td> x</tr>
<tr> <th> Математические выражения <td> :<td> :<td> x <td> x </tr>
<tr> <th> Панели инструментов <td> :<td> :<td> x <td> x </tr>

<tr> <th> Таблицы <td> :<td> :<td> x <td> x </tr>
<tr> <th> Доступность <td> :<td> :<td> :<td> x </tr>
<tr> <th> Внедренные объекты <td> :<td> :<td> :<td> x </tr>
<tr> <th> Скрипты <td> :<td> :<td> :<td> x </tr>
</table>
</body>
</html>
```

С помощью тега <caption> можно задать надпись над таблицей. Каждая строка таблицы начинается с тега <tr> (table row — строка таблицы). Отдельные ячейки помечаются тегом <th>

(table header — заголовок таблицы) или <td> (table data — данные таблицы). Эти два тега позволяют создавать, соответственно, строки заголовков таблицы и обычные строки, что и показано в примере.

В таблицах также могут использоваться другие теги. С их помощью можно устанавливать параметры выравнивания содержимого ячеек по вертикали и горизонтали, задавать параметры границ ячеек, объединять ячейки в группы и многое другое.

HTML 4,0 отличается от предыдущих версий некоторыми новыми свойствами. Они включают в себя специальные методы доступа для людей с ограниченными возможностями, внедрение объектов (обобщение тега <img>, позволившее включать в состав страниц не только изображения, но и другие объекты), поддержку языков написания сценариев (скриптов), что дало толчок к развитию динамических страниц, и т.д.

Таблица 5.2 — Некоторые различия версий HTML

| Аспект | HTML 1.0 | HTML 2.0 | HTML 3.0 | HTML 4.0 |
|---|---|---|---|---|
| Гиперссылки | X | X | X | X |
| Изображения | X | X | X | X |
| Списки | X | X | X | X |
| Активные карты и изображения | | X | X | X |
| Формы | | X | X | X |
| Математические выражения | | | X | X |
| Панели инструментов | | | X | X |
| Таблицы | | | X | X |
| Доступность | | | | X |
| Внедренные объекты | | | | X |
| Скрипты | | | | X |

Если веб-сайт достаточно велик по размерам и сложен настолько, что над ним работает целая группа программистов, желательно обеспечить более или менее схожий дизайн всех страниц. Эта проблема решается при помощи таблиц стилей. При их использовании отдельные страницы оформляются не реальными (физическими) стилями (например, курсивом или полужирным



шрифтом), а логическими. Среди них могут быть, например, <dn> (Определение), <ет> (Слабое выделение),  (Сильное выделение), <var> (Программная переменная). Логические стили определяются в таблицах стилей, ссылка на которые ставится в начале кода каждой страницы. Таким образом, можно придать всем страницам единый вид. Если веб-мастер решит изменить стиль  и отображать его полужирным шрифтом величиной 18 пунктов радикального розового цвета вместо курсива величиной 14 пунктов синего цвета, ему необходимо будет лишь поменять одно определение в таблице стилей, чтобы изменения распространились на весь сайт. Таблицы стилей можно сравнить с файлами в языке С, включаемыми с помощью директивы #include, — там также изменение одного макроопределения влечет за собой изменение во всех программных файлах, использующих данный заголовочный файл.

*Формы*

Первая версия языка HTML фактически обеспечивала лишь одностороннюю связь. Пользователи могли получать страницы от поставщиков информации, но отправлять информацию обратно было довольно трудно. По мере того, как все больше коммерческих организаций начали использовать Всемирную паутину, потребность в двустороннем трафике стала возрастать. Так, многим компаниям потребовалась возможность принимать заказы на продукцию с помощью своих веб-страниц. Производители программного обеспечения хотели продавать по сети свои программы, для чего требовалась возможность заполнения регистрационных карточек. Компаниям, предоставляющим поиск информации в Паутине, было нужно, чтобы пользователи могли ввести ключевые слова поиска.

В результате в HTML 2.0 были включены формы. Формы могут содержать кнопки и поля для ввода текста, позволяющие пользователям делать выбор или вводить необходимую информацию, которую затем можно отсылать владельцу страницы. Для этой цели используется тег <input>. У него могут быть различные параметры, определяющие размер, назначение и другие свойства отображаемого окна. Наиболее часто используемыми формами являются пустые поля для ввода текста, флажки, пере-

ключатели, активные карты и кнопки подтверждения. В примере, приведенном в листинге 5.5, показаны некоторые из перечисленных форм. Отображение этой HTML-страницы браузером показано на рис. 5.9.

Листинг 5.5 HTML-текст для бланка заказа

```
<html>
<head> <title> Бланк заказа клиента awi </title> </head>
<body>
<hl> Бланк заказа штуковины </hl>
<form action=”http://www.widget.com/cgi-bin/widgetorder” method=post>
Имя <input name=”customer” size=60> <p>
Адрес <input name=”address” size=58> <p>
Город <input name=”city” size=21>
Штат <input name=”state” size=4>
Страна <input name=”country” size=10> <p>
№ кредитной карты <input name=”cardno” size=10>
Срок действия <input name=”expires” size-4>
m/c <input name “cc” type=radio value=”mastercard”>
visa <input name =”cc” type=radio value=”visacard”> <p>
Размер штуковины: большой <input name=”product” type=radio value=”дорогая”>
маленький <input name=”product” type=radio value=”дешевая”>
Доставка <input name=”express” type=checkbox> <p>
<input name=submit value=”отправить заказ”> <p>
Благодарим вас за то, что вы заказали у нас штуковины фирмы AWI. Это правильный выбор!
</form>
</body>
</html>
```

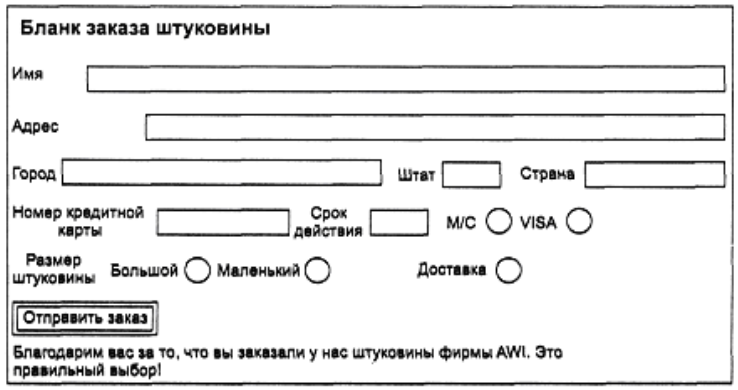



Рисунок 5.9 — Форматированная страница



Начнем изучение форм с этого примера. Как и все формы, она заключена между тегами <form> и </form>. Текст, не заключенный в теги, просто отображается. Внутри формы разрешено использование всех обычных тегов (например, <b>). В данной форме используются три типа окон для ввода данных.

Окно первого типа следует за текстом «Имя». Ширина этого окна 46 символов. Предполагается, что пользователь введет здесь свое имя, которое будет храниться в виде текстовой строки в переменной customer для последующей обработки. Тег <p> указывает браузеру, что последующий текст следует отображать с новой строки, даже если в текущей строке еще достаточно места. С помощью этого и других тегов автор страницы может управлять внешним видом бланка на экране.

В следующих окнах формы спрашивается адрес заказчика, то есть улица, город, штат и страна. Между этими полями не вставляются теги <p>, поэтому браузер по возможности пытается отобразить их все в одной строке. С точки зрения браузера, этот абзац представляет собой просто шесть отдельных элементов — три строки, перемежающиеся тремя окнами. Он отображает их друг за другом, слева направо, переходя на новую строку по мере необходимости. Таким образом, вполне возможно, что на экране с разрешением 1600x1200 точек все три строки и соответствующие им окна поместятся в одну строку, тогда как на экране с разрешением 1024x768 точек они будут разбиты на две строки. В худшем случае слово «Страна» может оказаться в конце одной строки, а соответствующее окно ввода — в начале следующей строки.

В следующей строке у пользователя запрашивается номер кредитной карты и срок ее действия. Передавать номера кредитных карт по Интернету следует только в том случае, если приняты все соответствующие меры предосторожности.

Следом за датой истечения срока действия кредитной карты мы обнаруживаем новые для нас элементы управления — переключатели. Они используются, когда требуется выбрать только один вариант из нескольких. Они напоминают кнопки на автомагнитолах, служащие для быстрого доступа к заданным радиостанциям. Браузер отображает эти элементы управления таким образом, что пользователь может включать их щелчком мыши

на них (или с помощью клавиатуры). Переключатели этого типа всегда объединяются в группы. При этом включение одного переключателя автоматически выключает все остальные переключатели этой группы. Внешний вид переключателя зависит от используемого графического интерфейса. В пункте бланка «Размер штуковины» также используются два переключателя, Группы переключателей определяются по значению параметра name (имя) тега <input>. Специальных скобок из тегов вроде <radiobutton> .. </radiobutton > для определения групп переключателей не предусмотрено.

Параметр value указывает, какая кнопка была нажата пользователем. В зависимости от того, какую кредитную карту выберет пользователь для своих расчетов, переменной ее будет присвоено значение текстовой строки «mastercard» или «visacard».

Следом за двумя наборами переключателей в бланке используется элемент управления типа checkbox (флажок). Он похож на переключатель предыдущего типа, так как тоже может находиться в одном из двух состояний (установлен/ сброшен), но не объединяется в группы и включается и выключается щелчком мыши на нем независимо от других элементов управления того же типа. Например, заказывая пиццу на веб-странице компании Electropizza, пользователь может выбрать и сардины, и лук, и ананас (если у него крепкий желудок), но ему не удастся одновременно выбрать маленькую, среднюю и большую пиццу. Выбор приправы к пицце будет представлен в виде трех отдельных элементов управления типа checkbox, тогда как выбор размера пиццы будет реализован с помощью набора переключателей.

Если список, из которого предстоит сделать выбор пользователю, очень длинный, то использование переключателей несколько неудобно. В этом случае можно использовать теги <select> и </se1ect> для определения списка альтернатив. При этом семантика переключателей сохраняется, если только не используется параметр multiple, превращающий список альтернатив в набор независимо устанавливаемых и сбрасываемых флажков. Некоторые браузеры отображают пункты списка между тегами <select> и </select> как всплывающее меню.



Итак, мы рассмотрели два встроенных типа тега <1nput>: radio и checkbox. На самом деле, мы уже познакомились и с третьим типом этого тега, то есть с типом text. Мы не заметили этого, потому что этот тип является типом по умолчанию, поэтому параметр type = text можно не указывать. Еще два возможных значения параметра type: password и textarea. Окно password аналогично окну text, с той разницей, что при вводе текста в окне password символы не отображаются на экране. Окно textarea отличается от окна text тем, что может содержать несколько строк текста.

Возвращаясь к примеру на рис. 5.9, мы, наконец, добираемся до последнего использованного в этом примере элемента управления — кнопки submit (подтверждение). Когда пользователь щелкает мышью на этой кнопке, заполненный им бланк отсылается обратно на компьютер, на котором размещена эта веб-страница. Как и все остальные рассмотренные типы, submit является зарезервированным словом, интерпретируемым браузером. Строка параметра value (значение) в данном случае содержит надпись на кнопке. В принципе, все элементы управления, образуемые с помощью тега <input>, могут иметь параметр value. В окнах ввода текста содержимое параметра value отображается в окне редактирования, и пользователь может редактировать или удалить его. Элементы управления checkbox и radio также могут быть инициализированы с помощью специального служебного слова checked («выбрано»). Дело в том, что value просто отображает текст, но не отображает предпочитаемый выбор.

Когда пользователь нажимает кнопку submit, браузер упаковывает всю собранную информацию в одну большую строку и отправляет ее на сервер для обработки. Поля с данными разделяются амперсандами (&), а вместо пробелов ставятся знаки +. В нашем примере такая строка, отсылаемая на сервер, может выглядеть так, как показано ниже (вы видите три строки, а не одну только из-за недостаточной ширины бумажного листа):

customer-John+Doe&address-100+Ma1n+St.&dty-White+Pla1ns&state-NY&country-USA&cardno-1 234567890&expires-6/98&cc-mastercard&product-cheap&express-on

Это сообщение отправляется на сервер в виде одной текстовой строки. Если флажок элемента управления checkbox сброшен, соответствующая ему переменная опускается. Сервер сам решает, что ему делать с полученной строкой.

*XML и XSL*

Язык HTML — с формами или без оных — никак не определяет структуру вебстраниц. Он смешивает содержимое страницы и описание средств ее форматирования. По мере роста популярности электронной коммерции и других приложений появлялась все более очевидная необходимость в структурировании веб-страниц и отделении содержимого от форматирования. Например, поисковая программа, обещающая найти в Мировой паутине книгу или компакт-диск по самой выгодной цене, должна проанализировать множество страниц, находя нужное наименование и цену. Если страница написана на обычном HTML, такой программе будет очень тяжело определить, где указано название товара, а где — его цена. По этой причине консорциум WWW (W3C) предложил расширение HTML, позволяющее структурировать страницы для облегчения их автоматической обработки. Для целей было создано два языка. Первый, XML (extensible Markup Language — Расширяемый язык разметки веб-страниц), описывает структурированное содержимое страниц, а второй, XSL (extensible Style Language — расширяемый язык стилей), описывает форматирование независимо от содержимого. И о том, и о другом можно говорить очень долго, поэтому нам приходится ограничиться лишь поверхностным описанием идей, лежащих в основе этих языков.

Рассмотрим документ XML, представленный в листинге 5.6. В нем определяется структура book_list, представляющая собой список книг. Под каждую книгу отведено три поля: название, автор и год издания. Эти структуры чрезвычайно просты. Разрешается иметь структуры с повторяющимися полями (например, несколько полей с именами авторов), необязательными полями (например, название прилагающегося компакт-диска), а также альтернативные поля (например, URL магазина, если книга еще есть в продаже, и URL аукциона, если весь тираж уже распродан).



В приведенном примере каждое поле является неделимой сущностью, однако разрешается разделять поля на подполя. Например, поле, содержащее имя автора, может быть — для улучшения возможностей поиска и форматирования — организовано следующим образом:

```
<author>
<first_name> Эндрю </irst_name>
<last_name> Таненбаум </last_name>
</author>
```

Итак, любое поле может иметь подполя неограниченной вложенности.

Код, представленный в листинге 5.6, делает лишь одно: определяет список из трех книг. Ничего не говорится о том, как должна выглядеть веб-страница на экране. Информация о форматировании страницы берется из другого файла, book_ list.xsl, содержащего определения XSL. Реально данный файл представляет собой таблицу стилей, в которой оговаривается вид страницы. (Существуют и альтернативы таблицам стилей, позволяющие, например, преобразовывать XML в HTML, однако обсуждение этой темы выходит за рамки этой книги.)

Листинг 5.6 Простой пример на XML

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="book_list.xsl" ?>
<book_list>

<book>
<title> Программное обеспечение сетей ЭВМ </title>
<author> И. В. Бойченко </author>
<year> 2005 </year>
</book>

<book>
<title> Архитектура вычислительных систем </title>
<author> И. В. Бойченко </author>
<year> 2004 </year>
</book>

<book>
<title> Сети ЭВМ и телекоммуникации </title>
<author> И. В. Бойченко </author>
<year> 2003 </year>
```

```
</book>

</book_list>
```

Пример XSL-файла для форматирования страницы из листинга 5.6 приведен в листинге 5.7. За некоторыми необходимыми объявлениями, включающими, например, URL используемого стандарта XSL, следуют теги, первыми из которых являются <html> и <body>. С этого начинается любая обычная вебстраница. Затем следует определение таблицы, включающее заголовки трех столбцов. Обратите внимание на то, что в дополнение к тегам <th> поставлены закрывающие теги </th>. Раньше нам было все равно, есть они или нет. Однако спецификации XML и XSL куда строже, чем HTML. Оговаривается, что страницы с синтаксическими ошибками должны отвергаться браузерами в любом случае, даже если они в состоянии понять, что имел в виду разработчик страницы. Браузер, отображающий синтаксически некорректный код XML или XSL, будет сам по себе признан некорректным при первом же тестировании на совместимость со стандартами. Однако браузерам разрешается выявлять ошибочные места. Такие драконовские меры нужны для борьбы с несметным числом небрежно написанных страниц, которые появились в Сети за последние годы.

Листинг 5.7 Таблица стилей на XSL

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">

<html>
<body>
<table border="2">
<tr>
<th> НАЗВАНИЕ </th>
<th> АВТОР </th>
<th> ГОД </th>
</tr>

<xsl:for-each select="book_list/book">
```



```
<tr>
<td> <xsl:value-of select="title"/> </td>
<td> <xsl:value-of select="author"/> </td>
<td> <xsl:value-of select="year"/> </td>
</tr>
</xsl:for-each>
</table>
```

Краткие пояснения к листингу 5.7. Выражение

```
<xsl: for-each select=”book_list/book”>
```

аналогично подобному выражению на языке С. С его помощью запускается цикл (ограниченный тегами <xsl :for-each>). На каждую книгу приходится одна итерация этого цикла. И каждая итерация выдает пять строк: <tr>, название, автор, год и тег </tr>. По окончании цикла выводятся закрывающие теги </body> и </html>. Результат интерпретации браузером этой таблицы стилей такой же, как если бы это была обычная страница, содержащая таблицу. Однако благодаря такому формату анализирующая программа сможет по XML-файлу легко найти, например, книги, изданные после 2000 года. Надо отметить, что, хотя наш XSL-файл сдержит нечто вроде цикла, веб-страницы на XML и XSL все равно остаются статическими, поскольку они содержат лишь инструкции, указывающие браузеру, как отображать страницу. Тем же, в принципе, занимается и HTML. Разумеется, чтобы интерпретировать XML и XSL, браузер должен поддерживать эти языки. На сегодняшний день, впрочем, большинство браузеров имеют такую возможность. До сих пор не очень понятно, заменит ли XSL традиционные таблицы стилей.

Мы не показали этого в нашем примере, но XML позволяет разработчику вебстраницы определять структуры заранее в специальном файле. Такие файлы определений затем можно подключать для построения сложных страниц. Дополнительную информацию, касающуюся этого и многих других свойств XML и XSL, можно найти в любой из многочисленных книг, посвященных этой теме.

Перед тем как закончить обзор об XML и XSL, сделаем замечание об идеологической борьбе между консорциумом WWW

и сообществом веб-дизайнеров. Изначальной целью HTML было определение именно структуры документа, а вовсе не его внешнего вида. Например, строка сообщает браузеру о том, что следует выделить заголовок, однако ничего не говорит о его гарнитуре, размере или цвете. Все эти детали реализует браузер по своему усмотрению: у него есть преимущество, состоящее в том, что он знает свойства конкретного монитора (например, сколько на нем точек). Однако дизайнерам веб-страниц в какой-то момент захотелось получить тотальный контроль над видом создаваемых страниц. Тогда были добавлены новые теги, уточняющие, как должен выглядеть документ. Например, были добавлены методы точного позиционирования элементов на экране. Проблема, присущая такому подходу, заключается в том, что такие страницы не обладают свойством переносимости. Они могут замечательно смотреться на браузере у создателя, однако на другом браузере, другой версии того же браузера или просто на экране с другим разрешением могут выглядеть совершеннейшей кашей. Одна из задач XML состояла в попытке вернуться к истокам, когда определялась только структура, а не внешний вид документа. Вместе с тем, XSL позволяет управлять тем, как выглядят страницы. Оба языка, впрочем, порой используются не по назначению. Следует иметь это в виду.

XML можно использовать не только для описания веб-страниц. Все чаще он используется в качестве языка для связи между прикладными программами. В частности, SOAP (Simple Object Access Protocol — простой протокол доступа к объектам) предоставляет возможность выполнения удаленных вызовов процедур между приложениями способом, независимым от языка и системы. Клиент формирует запрос в виде сообщения XML и отсылает его на сервер по описываемому далее протоколу HTML. Сервер отсылает назад ответ, представляющий собой форматированное XML-сообщение. Таким образом, могут общаться приложения, работающие на разнородных платформах.

*XHTML — расширенный язык разметки гипертекста*

К языку HTML постоянно предъявляются новые требования. Многие представители этой индустрии чувствуют, что в будущем большинство устройств, связанных со Всемирной паутиной, будут представлять собой не ПК, а беспроводные портативные устройства типа PDA. У таких мини-компьютеров нет столь большого объема памяти, чтобы работать с большими браузерами, обладающими сложной эвристикой, с помощью которой они пытаются отображать синтаксически некорректные страницы. Таким образом, следующей версией после HTML 4 должен стать язык, отличающийся крайне высокой требовательностью. Он называется не HTML 5, а XHTML, поскольку, по сути дела, представляет собой HTML 4, приведенный к стандарту XML. Под этим мы подразумеваем, что теги типа <hl> не имеют существенного значения. Чтобы добиться от такого тега того эффекта, который он производит в HTML 4, необходимо написать определение на XSL в отдельном файле. XHTML — это новый веб-стандарт, который рекомендуется использовать при создании любых веб-страниц для достижения максимальной переносимости на другие платформы и независимости отображения от браузера.



Между XHTML и HTML 4 существует шесть существенных и множество мелких различий. Во-первых, страницы и браузеры стандарта XHTML должны работать в строгом соответствии со стандартом. Низкопробные страницы уже отжили свой век. Это свойство унаследовано из XML.

Во-вторых, все теги и атрибуты должны быть написаны строчными буквами. Так, тег <HTML> будет считаться некорректным в XHTML. Необходимо писать <html>. Аналогично, некорректной записью считается такая: <1mg SRC=«pic001.jpg»> Она содержит имя атрибута, написанное заглавными буквами, а это запрещено.

В-третьих, всегда должны присутствовать закрывающие теги, даже для </p>. Если у тега не может быть естественного закрывающего тега (например, <br>, <hr>, <img>), то перед закрывающей скобкой тега следует ставить косую черту, Например,

<img src=«pic001.jpg» />

В-четвертых, все значения атрибутов должны указываться в двойных кавычках. Вот пример неправильного использования тега:

<img src=«pic001.jpg» height=500 />

Число 500 должно быть заключено в двойные кавычки, как и имя JPEG-файла.

В-пятых, свойство вложенности тегов должно использоваться корректно. В прошлом это не требовалось, важно было только получить ожидаемый результат. Раньше вполне легальным было написать:

  Летние фотографии  

В XHTML это запрещено. Закрывающие теги должны быть написаны строго в обратном порядке по отношению к открывающим тегам.

В-шестых, в каждом документе должен быть указан его тип. Мы имели возможность в этом убедиться на примере листинга 5.7. Все серьезные и мелкие изменения, которые происходят в стандартах, обсуждаются на сайте www.w3.org.

### 5.2.3 Динамические веб-документы

Все идеи, рассматривавшиеся до сих пор, соответствуют модели, показанной в листингах 4.2, 4.3: клиент сообщает серверу имя файла, а тот в ответ возвращает файл. В первые годы существования Всемирной паутины все ее содержимое и в самом деле было статическим (просто файлы). Однако в последние годы в Сети появляется все больше динамических объектов, то есть таких, которые создаются по требованию, а не хранятся постоянно на диске. Автоматическое создание объектов может происходить как на стороне сервера, так и на стороне клиента. Рассмотрим оба случая по порядку.

*Динамическая генерация содержимого веб-страниц на стороне сервера*

Чтобы понять, зачем вообще нужна динамическая генерация веб-страниц на стороне сервера, рассмотрим использование форм, описанных ранее. Когда пользователь заполняет поля формы и нажимает кнопку *Submit* (Подтверждение), серверу отправляется сообщение, содержащее в себе данные, предоставленные пользователем. Это сообщение не содержит имя запрашиваемого файла. Требуется, чтобы оно было передано для обработки программе или скрипту. Обычно обработка подразумевает использование пользовательских данных для поиска по базе данных на серверном диске и создание HTML-страницы, содержимое которой зависит от результатов этого поиска. Затем страница отсылается клиенту. Например, в приложении для электронной коммерции нажатие кнопки *ПЕРЕЙТИ К РАСЧЕТАМ* приводит к тому, что браузер возвращает на сервер cookie-файл с содержимым корзины клиента. На сервере при этом должны запуститься определенные программы или скрипты, в задачу которых входят обработка cookie и генерация HTML-страницы. Отправляемая клиенту HTML-страница может содержать форму со списком товаров, положенных в корзину, и адрес доставки вместе с запросом подтверждения заказа и предложением выбрать одну из возможных форм оплаты. Этапы обработки информации, полученной из HTML-формы, показаны на рис. 5.10.



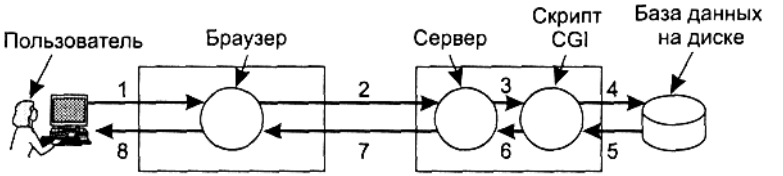



Рисунок 5.10 — Этапы обработки информации, полученной из формы

Традиционный способ работы с формами и другими видами интерактивных вебстраниц связан с использованием системы CGI (Common Gateway Interface — общий шлюзовой интерфейс). Это стандартизованный интерфейс, позволяющий вебсерверам общаться с прикладными программами и скриптами, разрешающими вводить данные (например, в формы) и в ответ генерировать HTML-страницы. Обычно такие прикладные программы представляют собой скрипты, написанные на языке описания сценариев Perl, поскольку писать их проще и быстрее, чем программы (по крайней мере, если вы умеете программировать на Perl). Существует договоренность, в соответствии с которой эти скрипты должны размещаться в каталоге CGI-BIN,

который доступен при помощи URL. Иногда вместо Perl используется другой язык написания скриптов, Python.

В качестве примера работы CGI рассмотрим случай, когда продукция компании «Великие Штуковины» приходит без гарантийного талона. Вместо этого клиенту предлагается зарегистрироваться в Интернете по адресу www.grwd.com. Там имеется ссылка:

«Щелкните здесь для регистрации приобретения»

Ссылка эта может указывать, например, на сценарий на языке Perl, расположенный по адресу www.grwd.com/cgi-bin/reg.perl. При запуске этого сценария без параметров обратно отсылается HTML-страница, содержащая регистрационную форму. Когда пользователь заполняет ее и нажимает кнопку *Submit,* скрипту передается сообщение, содержащее указанные им значения. Вид этого сообщения традиционен при работе с формами. Что происходит дальше, предугадать нетрудно. Perl-скрипт анализирует параметры, создает в базе данных запись о новом клиенте, отсылает назад HTML с регистрационным номером и телефоном службы поддержки. Понятно, что это не единственный способ обработки форм, однако он является наиболее распространенным. О создании CGI-скриптов и программировании на Perl написано много книг. Среди них стоит отметить (Hanegan, 2001; Lash, 2002; Meltzer и Michalski, 2001).

Динамическое создание веб-страниц на стороне сервера может быть реализовано не только с помощью CGI-скриптов. Существует еще один распространенный способ, который заключается во внедрении небольших скриптов в HTML-страницы. Они выполняются на сервере, в их задачу входит генерирование страницы. Популярным инструментом для написания таких скриптов является PHP (Hypertext Preprocessor — гипертекстовый препроцессор). При его использовании требуется, чтобы сервер понимал PHP (точно так же, как браузер должен понимать XML, чтобы интерпретировать страницы, написанные на одноименном языке). Обычно серверы предполагают, что у файлов страниц, написанных на PHP, расширение php, а не htm или html.

В листинге 5.8 приведен пример маленького скрипта на PHP; он должен работать на любом сервере, если на нем установлен гипертекстовый препроцессор. Он состоит из привычного обрамления на HTML, а сам скрипт содержится внутри тега <?php ... ?>. Вся его работа заключается в создании страницы, сообщающей всю известную информацию о запустившем его браузере. Браузеры обычно отправляют кое-какие данные о себе вместе с запросами (и любыми прикладными cookie-файлами). Эти данные сохраняются в переменной HTTP_USER_AGENT. Если этот листинг сохранить в файле test.php в веб-каталоге компании «ABCD», то пользователь, набрав URL www.abcd.com/test.php, сможет получить страницу, из которой он узнает, какие браузер, язык и операционную систему он использует.



Листинг 5.8 Пример страницы на HTML с внедренным PHP-скриптом

```
<html>
<body>

<h2>А я вот что про тебя знаю:</h2>
<?php echo $HTTP_USER_AGENT ?>

</body>
</html>

<html>
<body>
<form action=”action.php” method=”post”>
```

PHP особенно хорошо подходит для обработки форм — с его помощью она осуществляется даже проще, чем путем написания CGI-скрипта. Пример обработки формы показан в листинге 5.9,*а.* Здесь мы видим обычную HTML-страницу с формой. Непривычно выглядит только первая строка, в которой указывается, что скрипт action.php должен быть запущен для обработки параметров после заполнения пользователем формы и нажатия кнопки подтверждения. Форма в этом примере состоит из двух текстовых полей ввода, в одном из которых запрашивается имя клиента, а в другом — его возраст. По окончании работы пользователя с формой на сервер отсылается стандартная строка, пример которой мы уже видели ранее. Эта строка обрабатывается, из нее извлекаются значения переменных name

и age. Затем начинает свою работу скрипт action.php, показанный в листинге 5.9,*б*. Он генерирует ответ. Работа скрипта заключается в исполнении php-команд. Если пользователь предоставил данные «Харриет» и «24», то ему будет прислан HTML-файл, код которого показан в листинге 5.9,*в*. Как видите, обработка форм с помощью PHP производится элементарно.

Несмотря на простоту использования, PHP — это мощный язык программирования, ориентированный на предоставление интерфейса между Всемирной паутиной и серверными базами данных. В PHP есть переменные, строки, массивы и большинство управляющих структур, присущих языку C, однако ввод-вывод гораздо мощнее, чем обычный printf. PHP имеет открытый исходный код и распространяется бесплатно.

Листинг 5.9 Веб-страница с формой (а); PHP-скрипт для обработки формы (б); результат работы PHP-скрипта при исходных данных «Харриет» и «24» соответственно (в)

```
<p> Введите свое имя: <input type=”text” name=”name”> </p>
<p> Введите свой возраст: <input type=”text” name=”age”> </p>
<input type=”submit”>
</form>
</body>
</html>
```

(а)

```
<html>
<body>
<hl> Ответ: </hl>
Привет. <?php echo $name: ?>!
Предсказываю: в следующем году тебе будет <?php echo $age+1: ?>
</body>
</html>
```

(б)

```
<html>
<body>
<hl> Ответ: </hl>
Привет, Харриет!
Предсказываю: в следующем году тебе будет 25
</html>
</body>
```

(в)



PHP был разработан специально под сервер Apache, который также обладает открытым исходным кодом и является самым распространенным веб-сервером в мире.

Итак, мы знаем уже два различных способа генерации динамических HTML-страниц: с помощью CGI-скриптов и внедрения PHP. Есть еще третий метод, называемый JSP (JavaServer Pages — страницы сервера Java). Он в целом схож с PHP и отличается только тем, что динамическая часть программируется на языке Java. Файлы страниц, написанных с помощью JSP, имеют одноименное расширение: .jsp. Еще один метод создания динамических страниц — ASP (Active Server Pages — активные серверные страницы). Это ответ Microsoft на PHP и JSP. В качестве языка динамического веб-программирования используется собственный язык написания скриптов, созданный Microsoft, — Visual Basic Script. Соответственно, файлы страниц, написанных с использованием этого метода, имеют расширение .asp. Вопрос выбора между PHP, JSP и ASP в основном политический (открытый код *Sun* против кода *Microsoft)*. С точки зрения технологий все эти методы вполне сравнимы по возможностям.

Весь набор методов создания динамических страниц иногда называют динамическим HTML (DHTML).

*Создание динамических веб-страниц на стороне клиента*

Скрипты CGI, PHP, JSP и ASP решают вопросы обработки форм и взаимодействия с базами данных, расположенными на сервере. Они могут принимать входящую информацию из форм, осуществлять поиск по одной или нескольким базам данных и в качестве результата генерировать HTML-страницы. Но ни один из этих методов не позволяет напрямую взаимодействовать с пользователем, например, реагировать на движения мышкой. Для этих целей необходимы скрипты, внедренные в HTML-страницы и выполняющиеся не на серверной, а на клиентской машине. Начиная с HTML 4.0, появилась возможность включать скрипты такого типа с помощью тега <script>. Наиболее популярный язык написания сценариев для клиентской стороны — это JavaScript.

JavaScript — это язык написания сценариев, использующий идеи, крайне отдаленно напоминающие язык программирования Java. Но JavaScript — это не Java по определению. Как и другие

языки написания скриптов, он очень высокоуровневый. Так, одной строкой JavaScript можно создать диалоговое окно, войти в цикл ожидания пользовательского ввода и сохранить полученную строку в переменной. Столь высокий уровень языка идеально подходит для разработки интерактивных веб-страниц. С другой стороны, тот факт, что JavaScript не стандартизован и мутирует быстрее, чем мушка-дрозофила в рентгеновском луче, сильно усложняет написание платформонезависимых программ. Надо, впрочем, надеяться, что рано или поздно этот язык дойдет до более или менее устойчивого состояния.

Пример программы на JavaScript показан в листинге 5.10. Как и в листинге 5.9,*а,* программа создает форму с запросом имени и возраста пользователя и гениальным образом предсказывает, исходя из этих данных, каков будет возраст человека в следующем году. Тело скрипта почти такое же, как и в примере PHP. Основная разница состоит в объявлении кнопки *Submit* и определении присваивания в этом объявлении. Оператор присваивания сообщает браузеру о том, что в случае нажатия кнопки необходимо запустить скрипт response и передать ему форму в качестве параметра.

Листинг 5.10 Применение JavaScript для обработки формы

```
<html>
<head>
<script language=”javascript” type=”text/javascript”>
function response (test_form) {
var person = test_form.name.value:
var years = eval (test_form.age.value) + 1:
document.open ()
document.writeln (“<html> <body>”):

document.writeln (“Привет. ˝ + person + ˝ !<br>”):
document.writeln (“Предсказываю: в следующем году тебе будет ” + years + ˝ . ˝):
document.writeln (“</body> </html>”):
document.close ():
}
</script>
</head>

<body>
<form>
Введите свое имя: <input type=”text” name=”name”>
<p>
Введите свой возраст: <input type=”text” name=”age”>
```



```
<p>
<input type=”button” value=”Подтверждение” onclick=”response (this.form)”>
</form>
</body>
</html>
```

Совершенно по-новому здесь объявляется функция response. Объявление находится в заголовке HTML-файла, который обычно хранит информацию о заголовках, цвете фона и т. п. Функция извлекает из формы значение поля name и сохраняет его в виде строки в переменной person. Также извлекается значение поля age. Оно приводится к целочисленному типу с помощью функции eval, затем к значению добавляется 1, и результат сохраняется в years. После этого документ открывается для записи, в него записываются четыре строки (для этого используется метод writeln), и документ закрывается. Документ представляет собой HTML-страницу, как видно по многочисленным тегам HTML. Браузер выводит готовый документ на экран.

Важно понимать, что обрабатываются программы, показанные в листингах 5.9 и 5.10, совершенно по-разному, несмотря на их внешнее сходство. Что происходит в первом случае (листинг 5.9)? После того как пользователь нажимает кнопку *Submit,* браузер собирает всю введенную информацию в одну длинную строку и отправляет ее на тот сервер, с которого пришла страница. Сервер видит имя PHP-файла и запускает его. PHP-скрипт создает новую HTML-страницу, которая отсылается браузеру для отображения. Что касается второго случая (листинг 5.10), то после нажатия кнопки *Submit* браузер сам выполняет действия функции JavaScript, содержащейся на странице. Вся работа производится локально, внутри браузера. С сервером никакого взаимодействия не осуществляется. Как следствие, результат появляется практически мгновенно, тогда как при использовании PHP задержка прибытия страницы с результатом может составлять несколько секунд. Разница между скриптами, работающими на стороне сервере и на стороне клиента, показана на рис. 5.11. Показаны шаги, выполняемые в каждом случае. В обоих случаях все эти этапы производятся после вывода формы на экран. Шаг 1 состоит в приеме данных от пользователя.

Затем следует их обработка, и вот здесь имеются значительные различия.

Эти различия вовсе не означают, что JavaScript лучше, чем PHP. Просто у них различные сферы применения. PHP (а с ним неявно и JSP, и ASP) применяется тогда, когда необходимо взаимодействовать с удаленной базой данных. JavaScript используют тогда, когда требуется взаимодействие с пользователем в пределах его компьютера. Разумеется, возможно (и довольно часто осуществляется) одновременное использование PHP и JavaScript, хотя они и не могут, например, обрабатывать одно и то же событие (типа нажатия кнопки) или производить одно и то же действие.

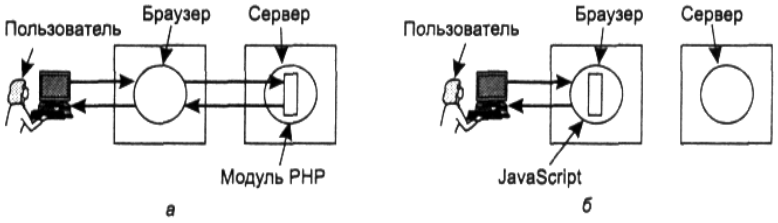



Рисунок 5.11 — PHP-скрипт на стороне сервера (а); сценарий на JavaScript на стороне клиента (б) (нумерация)

JavaScript — это полноценный язык программирования, ничуть не слабее по возможностям, чем C или Java. В нем есть понятия переменных, строк, массивов, объектов, функций и всех привычных управляющих структур. К тому же он обладает множеством полезных свойств, специфичных для веб-страниц, включая возможность управления окнами и фреймами, установки и получения cookie, работы с формами и гиперссылками. Пример программы на JavaScript с рекурсивной функцией приведен в листинге 5.11.

Листинг 5.11 Программа на JavaScript для подсчета и вывода факториалов

```
<html>
<head>
<script language=”javascript” type=”text/javascript”>

function response (test_form) {
```



```
function factorial (n) {if (n-o) return 1: else return n * factorial (n-1)}
var r=eval (test_form.number.value): // r=введенный аргумент
document.myform.mytext.value = “Результаты:\ n”:
for (var i = 1: I <=r : I ++)// Вывести одну строку от 1 до r
document.myform.mytext.value += (I + “!=” + factorial (1) + “\n”):
}
</script>
</head>

<body>
<form name=”myform”>
Ведите число: <input type=”text” name=”number”>
<input type=”button” value=”Подсчет таблицы факторилов”
onclick=”response (this.form)”>
<p>
<textarea name=”mytext” rows=”25” cols=”50”> </textarea>
</form>
</body>
</html>
```

С помощью JavaScript можно также отслеживать появление мыши над определенными объектами на странице. На многих веб-страницах с JavaScript можно заметить, что при наведении курсора мыши на какой-нибудь текст или изображение что-нибудь происходит. Часто при этом меняется изображение или вдруг выскакивает меню. Такое поведение легко программируется на JavaScript и несколько оживляет веб-страницы. Пример приведен в листинге 5.12.

Листинг 5.12 Интерактивная страница, отвечающая на движения «мышью»

```
<html>
<head>
<script language=”javascript” type=”text/javascript”>

if (!document.myurl) document.myurl = new Array ():
document.myurl [0] =”http://www.cs.vu.nl/ast/im/kitten.jpg”:
document.myurl [1] =”http://www.cs.vu.nl/ast/im/puppy.jpg”:
document.myurl [0] =”http://www.cs.vu.nl/ast/im/bunny.jpg”:
function pop (m) {
var urx=”http://” www.cs.vu.nl/ast/im/cat.jpg”:
popuwin=window.open (document.myurl [m]. “mywind”. “wight=250.height=250”):
}
</script>
</head>

<body>
<p><a href=”#” onMouseover=”pop (0): return false:”> Котенок </a></p>
<p><a href=”#” onMouseover=”pop (1): return false:”> Щенок </a></p>
```

```
<p><a href="#" onMouseover="pop (2): return false:"> Кролик </a></p>
</body>
</html>
```

JavaScript — это не единственный инструмент для создания веб-страниц с высокой степенью интерактивности. Еще один метод связан с использованием апплетов. Это небольшие программы на Java, скомпилированные в машинные инструкции виртуального компьютера, называемого JVM (Java Virtual Machine — виртуальная машина Java). Апплеты могут внедряться в HTML-страницы (между тегами <applet> и </app1et>) и интерпретироваться JVM-совместимыми браузерами. Поскольку перед выполнением Java-апплеты проходят стадию интерпретации, интерпретатор может помочь избежать выполнения «нехороших действий». По крайней мере, теоретически такая возможность существует. На практике создатели апплетов обнаружили почти бесконечный поток ошибок в библиотеках ввода-вывода Java.

Ответ корпорации Microsoft на Java-апллеты фирмы Sun состоял в подключении к веб-страницам управляющих элементов ActiveX — программ, скомпилированных для машинного языка процессора Pentium и выполняемых на аппаратном уровне. Это свойство делает их значительно более быстрыми и гибкими, нежели интерпретируемые Java-апплеты, поскольку они могут делать все то же самое, что и обычная программа. Когда Internet Explorer видит на странице управляющий элемент ActiveX, он загружает его, идентифицирует и исполняет. Однако загрузка инородных программ может породить проблемы защиты информации. Поскольку почти все браузеры могут интерпретировать программы как на Java, так и на JavaScript, разработчик, желающий создать страницу с высокой степенью интерактивности, может выбирать, по крайней мере, из двух этих технологий. А если вопрос платформонезависимости значения для него не имеет, то к ним добавляется еще и ActiveX. Общее правило таково:

+ JavaScript обеспечивает простоту программирования.
+ Java-апплеты обеспечивают быстроту выполнения.



+ Управляющие элементы ActiveX обгоняют по скорости выполнения все остальные технологии.

Во всех браузерах реализована строго одна и та же версия JVM, а вот найти два браузера с одинаковой реализацией JavaScript почти невозможно. Поэтому Java-апплеты легче переносятся с платформы на платформу, чем JavaScript. Существует множество увесистых (часто более 1000 страниц) книг, посвященных JavaScript.

Прежде чем закончить тему динамических веб-страниц, подведем краткие итоги изученного. Целые веб-страницы могут создаваться «на лету» с помощью разнообразных скриптов, работающих на сервере. С точки зрения получающего их браузера, это обычные HTML-страницы, которые нужно просто вывести на экран. Скрипты можно создавать посредством технологий Perl, PHP, JSP или ASP (рис. 5.12).


Рисунок 5.12 — Различные способы создания и вывода динамических страниц

Генерация динамических страниц возможна и на стороне клиента. Веб-страницы можно писать на XML и затем конвертировать в HTML, задавая внешнее представление в XSL-файле. Программы на JavaScript могут производить любые вычисления. Наконец, подключаемые модули и вспомогательные приложения могут использоваться для отображения самых разнообразных форматов.

## СПИСОК ЛИТЕРАТУРЫ

1. Таненбаум Э. Компьютерные сети. — 4-е издание. — М: Питер, 2003.

2. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. http://www.citforum.ru/operating_systems/sos/contents.shtml

3. Зайцев С.С., Кравцунов М.И., Ротанов С.В. Сервис открытых информационно-вычислительных сетей: Справочник. — М: «Радио и связь», 1990.

4. Храмцов П. Б., Брик С.А., Русак А.М., Сурин А.И. Основы web-технологий Интернет-университет информационных технологий — ИНТУИТ.ру (http://www.intuit.ru)

5. Храмцов П.Б., Брик С.А., Русак А.М., Сурин А.И. Введение в HTML технологии Интернет-университет информационных технологий — ИНТУИТ.ру. (http://www.intuit.ru)

6. Храмцов П.Б., Брик С.А., Русак А.М., Сурин А.И. Введение в JavaScript Интернет-университет информационных технологий — ИНТУИТ.ру. (http://www.intuit.ru)