Обзорная лекция по курсу «Сети ЭВМ» для специальности Т10.02

Концепция сокетов.

Интерфейс прикладной программы (API) представляет собой просто набор функций (интерфейс), использующийся программистами для разработки прикладных программ в определенных компьютерных средах. Программист Windows, например, желающий написать программу, может воспользоваться широким набором функций, предоставленных в их распоряжение фирмой Microsoft - интерфейсом прикладных программ Windows или Windows API. В восьмидесятых годах американское правительственное агентство по поддержке исследовательских проектов (ARPA), финансировало реализацию протоколов TCP/IP для UNIX в Калифорнийском университете в г. Беркли. В ходе этого проекта группа исследователей-программистов разработала интерфейс прикладного программирования для сетевых приложений TCP/IP (TCP/IP API). По причинам, изложенным ниже, этот интерфейс был назван *сокетами* TCP/IP (TCP/IP sockets).

Так как изначально разработчиками интерфейса сокетов были исследователи университета в г. Беркли, этот интерфейс часто называется сокетами Беркли (Berkeley sockets) или сокетами в стиле Беркли.

Интерфейс сокетов - это API для сетей TCP/IP. Другими словами, он описывает набор программных функций или процедур, позволяющий разрабатывать приложения для использования в сетях TCP/IP. В наши дни интерфейс сокетов наиболее широко используется в сетях на базе TCP/IP.

Абстракция сокетов

Сокет можно рассматривать, как конечный пункт передачи данных по сети. Сетевое соединение — это процесс передачи данных по сети между двумя компьютерами или процессами. Сокет — конечный пункт передачи данных. Другими словами, когда программы используют сокет, для них он является абстракцией, представляющий одно из окончаний сетевого соединения. Для установления соединения в абстрактной модели сокетов необходимо, чтобы каждая из сетевых программ имела свой собственный сокет. Связь между двумя сокетами может быть ориентирована на соединение, а может быть и нет. Несмотря на то, что разработчики модифицировали системный код UNIX, интерфейс сокетов по-прежнему использует концепцию ввода-вывода данных UNIX, то есть сетевая модель интерфейса сокетов до сих пор использует цикл открыть-считать-записать-закрыть.

Чтобы открыть или создать файл в UNIX (и в большинстве других ОС), вы должны указать его описание (например, имя файла и то, как вы будете его использовать: записывать или считывать). Затем вы запрашиваете у операционной системы дескриптор файла, соответствующий вашему описанию. Не существует каких-либо ограничений на то, когда запрашивать дескриптор. Как только вам нужен файл, запрашивайте его дескриптор. В один и тот же момент времени может быть открыто несколько устройств или файлов. В любом случае операционная система возвращает дескриптор (обычно это целое число), однозначно соответствующий указанному файлу или устройству. Интерфейс сокетов работает точно так же. Когда программе нужен сокет, она формирует его характеристики и обращается к API, запрашивая у сетевого программного обеспечения его дескриптор. Структура таблицы с описанием параметров сокета весьма незначительно отличается от структуры таблицы с описанием параметров файла. Однако это отличие важно.

Отличие дескриптора сокета от дескриптора файла

Процессы получения дескриптора файла и сокета от операционной системы отличаются незначительно. Однако таблицы, на которые указывают дескрипторы, отличаются между собой. Тогда как дескриптор файла указывает на определенный файл (уже существующий или только что созданный) или устройство, дескриптор сокета не содержит каких-либо определенных адресов или пунктов назначения сетевого соединения. Тот факт, что дескриптор сокета не представляет определенную сетевую точку входа (endpoint), существенно отличает его от любого другого дескриптора стандартной системы ввода-вывода. В большинстве операционных систем дескриптор файла указывает на определенный файл, находящийся на жестком диске. Программы, работающие с сокетами, сначала образуют сокет и только потом соединяют его с точкой назначения на другом конце сетевого соединения. Если бы файловый ввод-вывод состоял из этих же шагов, приложение сначала получало бы дескриптор файла, а затем привязывало бы его к имени определенного файла на жестком диске.

Рассмотрим потребности сетевой программы TCP/IP, передающей информацию датаграммами по не ориентированному на соединение протоколу. Программа указывает адрес назначения датаграмм, но не устанавливает предварительного соединения с компьютером-получателем данных. Вместо этого программа передает датаграммы по адресу назначения. Сетевое программное обеспечение (уровень IP) обслуживает процесс доставки. Чтобы интегрировать TCP/IP в среду UNIX, разработчики сокетов должны были добавить к существующей системе ввода-вывода новые возможности. API сети TCP/IP был необходим способ получать дескриптор ввода-вывода, не устанавливая предварительно соединения с удаленным компьютером. Вместо того чтобы модифицировать существующую систему ввода-вывода UNIX, разработчики сокетов создали новую функцию, которая и получила название “сокет” *(socket).*

Создание сокета

Создавая программу TCP/IP, необходимо иметь возможность пользоваться как ориентированными, так и не ориентированными на соединение протоколами. Интерфейс сокетов позволяет программам использовать оба этих типа протоколов. Однако процессы создания сокета и соединения сокета с компьютером-получателем происходят

раздельно. Чтобы создать сокет, программа вызывает функцию socket. Она, в свою очередь, возвращает дескриптор сокета, подобный дескриптору файла. Другими словами, дескриптор сокета указывает на таблицу, содержащую описание свойств и структуры сокета. Следующий пример показывает возможную форму вызова функции socket:

socket_handle = socket(protocol_family, socket_type, protocol);

Создавая сокет, вы указываете три параметра: группу, к которой принадлежит протокол, тип сокета и сам протокол. Первый параметр задает группу или семейство, к которому принадлежит протокол, например семейство TCP/IP. Второй параметр, тип сокета, задает режим соединения: датаграммный или ориентированный на поток байтов. Параметр “протокол” определяет протокол, с которым будет работать сокет, например TCP.

Параметры сокета

Протоколы TCP/IP были не единственными, интегрированными разработчиками сокетов в систему UNIX. Кроме TCP/IP, встроенный API обслуживает и некоторые другие протоколы. Конечно, разработка API изначально ориентировалась на TCP/IP, однако и другие сети не были забыты. Различные семейства протоколов появились благодаря концепции универсальности API сокетов.

Семейства протоколов и адресов

Первый параметр, указываемый в вызове функции socket, определяет группу или семейство, к которому принадлежит протокол. Такой группой может быть, например, семейство TCP/IP. Так как могут быть выбраны различные семейства протоколов, интерфейс сокетов может обслуживать несколько различных типов сетей одновременно. Фактически, интерфейс сокетов Беркли обслуживает семейства протоколов TCP/IP и семейство протоколов сетевых служб фирмы Ксерокс (XNS).

XNS является многоуровневой системой протоколов, похожей на TCP/IP. Разработанная в исследовательском центре фирмы Ксерокс в Пало-Альто, система XNS послужила прототипом для некоторых популярных сетевых протоколов, таких как Novell Netware, Banyan Vines и

Для указания группы протоколов в интерфейсе сокетов определены символьные константы (макроопределения). Символьная константа PF_INET, например, определяет семейство протоколов TCP/IP. Константы, определяющие другие семейства, также начинаются с префикса “PF_”. Константа PF_UNIX определяет семейство внутренних протоколов ОС UNIX, а PF_NS — семейство протоколов фирмы Ксерокс.

Семейства адресов тесно связаны с семействами протоколов. Форматы адресов различных сетей не одинаковы. Разработчики сокетов в полном соответствии с этим соображением еще больше обобщили интерфейс сокетов, реализовав для работы с различными сетями возможность обращения к различным семействам сетевых адресов. Символьные константы, указывающие на семейство адресов, начинаются с префикса “AF_”. Константа, обозначающая семейство адресов Интернет (TCP/IP), называется AF_INET. Константа AF_NS обозначает семейство адресов фирмы Ксерокс, а AF_UNIX — файловой системы UNIX.

К сожалению, ввиду тесной взаимосвязи между семействами протоколов и семействами адресов, существует ошибочное мнение, что это одно и то же. К примеру, в интерфейсе сокетов TCP/IP значения констант семейства протоколов (PF_INET) и семейства адресов (AF_INET) равны. В результате, некоторые, замечательные в других отношениях справочные руководства, часто рекомендуют программистам использовать одно и то же значение в качестве обоих параметров вызова функции socket. Разница между семейством протоколов и семейством адресов позволяет программисту пользоваться несколькими семействами адресов внутри одного из семейства протоколов. Другими словами, интерфейс сокетов не накладывает ограничений на использование множества различных форматов адресов в рамках одного и того же семейства протоколов. Такая гибкость на самом деле не представляет особенного интереса для программистов Интернет, поскольку их “родной” протокол, TCP/IP, умеет пользоваться адресами только собственного формата. Несмотря на это досадное ограничение, сама гибкость подхода разработчиков демонстрирует планирование и предвидение, проявленные на стадии разработки интерфейса сокетов.

На сегодняшний день константы PF_INET и AF_INET имеют одинаковые значения, поэтому все равно, как их применять. Но если планировать на будущее, ситуация может измениться, и семейство адресов перестанет быть эквивалентным семейству протоколов. Если вы — программист Интернет, то лучше всего ставить константы там, где им положено находиться: PF_INET — для указания на семейство протоколов, а AF_INET — для указания на семейство адресов. Такой подход упростит понимание исходного текста программы и снимет потенциальные неприятности, могущие возникнуть при переносе программы в другую операционную систему.

Тип соединения

Соединения TCP/IP бывают двух режимов: ориентированные и не ориентированные на соединение. В ориентированных на соединение протоколах данные перемещаются как единый, последовательный поток байтов без какого-либо деления на блоки. В не ориентированных на соединение протоколах сетевые данные перемещаются в виде отдельных пакетов, называемых датаграммами. Как уже отмечалось, сокеты могут работать как с не ориентированными, так и с ориентированными на соединение протоколами. Второй параметр вызова функции socket обозначает тип соединения, который вы желаете использовать. Символьная константа SOCKJDGRAM обозначает датаграммы, а SOCK_STREAM — поток байтов.

Интерфейс сокетов также определяет третий тип соединения, называемый *“простой сокеты* (raw socket). Простой

сокет позволяет программе использовать напрямую те низкоуровневые протоколы, которые обычно используются сетевыми протоколами более высокого уровня. В главе 16 вы узнаете, что программа Ping создает простой сокет, чтобы напрямую использовать протокол управляющих сообщений Интернет (ICMP). Обычно ICMP служит для передачи сообщений о различных сетевых ошибках.

Как правило, прикладные программы не используют ICMP. Они предоставляют самой сети решать проблемы, связанные с ошибками. Транспортные протоколы Интернет самостоятельно доставят сообщение об ошибке, адресованное прикладной программе. Однако прикладная программа может напрямую обратиться к уровню IP или ICMP. Для этого ей придется создать простой сокет.

Выбор протокола

Семейство TCP/IP состоит из нескольких протоколов, например IP, ICMP, TCP и UDP. Любое семейство состоит из набора протоколов, которыми пользуются сетевые программисты. Третий параметр функции socket позволяет выбрать тот протокол, который будет использоваться вместе с сокетом. Как и в случае остальных параметров, протокол задается символьной константой.

В сетях TCP/IP все константы начинаются с префикса IPPROTO_. Например, протокол TCP обозначается константой IPPROTO_TCP. Символьная константа IPPROTO_UDP обозначает протокол UDP. Следующий оператор демонстрирует, как может выглядеть вызов функции socket:

```
socket_handle = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Данный вызов сообщает интерфейсу сокетов о том, что программа желает использовать семейство протоколов Интернет (PF_INET), протокол TCP (IPPROTO_TCP) для соединения, ориентированного на поток байтов (SOCK_STREAM).

Модель интерфейса сокетов

Парадигма сокетов, или модель интерфейса сокетов, рассматривает сетевые компьютеры в качестве конечных точек сетевого соединения. Каждое сетевое соединение включает две конечные точки: локальный компьютер и удаленный компьютер. В рамках интерфейса сокетов каждая конечная точка сети представлена сокетом. Вы знаете, что большинство сетевых программ пользуются моделью клиент-сервер. Сетевые соединения в рамках этой модели также включают две конечные точки соединения. Модель клиент/сервер делает эти две точки неравноправными. Одна должна выполнять функции сервера, а другая — клиента. Конечная точка-клиент инициирует запрос к сетевым службам и представлена программой-клиентом или процессом-клиентом. Конечная точка, отвечающая на запрос, представлена программой или процессом-сервером.

Сетевой уровень IP идентифицирует сетевые компьютеры при помощи сетевого адреса. Это значит, что каждый компьютер, подключенный к Интернет, должен иметь уникальный сетевой адрес. Каждый сетевой процесс, таким образом, использует номер порта сетевого компьютера в качестве собственного адреса. Наконец, вы знаете, что программы Интернет должны использовать семейство протоколов TCP/IP для передачи своих данных по сети. Таким образом, соединение между двумя сетевыми программами несет в себе следующую информацию:

• Местный (локальный) порт протокола, обозначающий программу или процесс, посылающий сообщения или датаграммы.

• Адрес локального компьютера, обозначающий сетевой компьютер, принимающий пакеты данных.

• Удаленный порт протокола, обозначающий программу или процесс-получатель данных.

• Протокол, обозначающий, каким образом программа собирается передавать данные по сети.

Таким образом, сокет является реализацией абстрактной модели конечной точки сетевого соединения. Структура данных сокета содержит все элементы, необходимые конечной точке сетевого соединения. Структура данных сокета значительно упрощает процесс сетевого соединения. Когда одна программа желает установить связь с другой, программа-передатчик просто отдает свою информацию сокету, а интерфейс сокетов в свою очередь передает ее дальше стеку сетевых протоколов TCP/IP. Перед этим программа должна создать сокет, вызвав системную функцию-сокет, а затем сконфигурировать его, пользуясь функциями, также входящими в интерфейс сокетов. В следующих разделах будет показано, каким образом сокет конфигурируется.

Использование сокета в программе

Перед тем как начать использовать сокет, программа должна сконфигурировать его. Структура его данных должна быть подготовлена к приему как номера порта, так и сетевого адреса компьютера. Термин *"адрес сокета"* относится не к самому сокету, а к адресам портов и компьютеров, размещенных внутри его структуры данных. Вызывая функцию socket, вы не указываете ни номер порта, ни сетевой адрес. Вместо этого вы передаете некоторые параметры. Вместе с протоколом указывается тип сетевой службы (ориентированный или не ориентированный на соединение). Также указывается, как будет работать программа: клиентом или сервером. (Назначение программы — выполнять функции клиента или сервера. Несмотря на то, что одна программа может быть и тем и другим, в один момент времени она может выполнять функцию либо только клиента, либо только сервера.) Параметры, передаваемые сокету, зависят от назначения программы, так же, как и от типа сетевой службы по доставке данных.

Соединение сокета

Сокет представляет абстракцию, пользуясь которой вы можете настраивать и программировать конечные точки

сетевого соединения. В предыдущей главе объяснялось, что ориентированные на соединение протоколы организуют между конечными точками виртуальную цепь. Другими словами, соединение между конечными точками в этом случае теоретически не отличается от выделенного двухточечного соединения. Транспортный уровень TCP/IP, TCP (ориентированный на соединение) обслуживает виртуальную цепь (держит соединение открытым), обмениваясь сообщениями-подтверждениями о доставке данных между двумя конечными точками. В результате, ориентированной на соединение программе-клиенту в сети TCP/IP нет дела до локального номера порта, с которого передаются ее данные. Программа-клиент может принимать данные на любом порту протокола. Поэтому в большинстве случаев ориентированные на соединение программы-клиенты не указывают номер локального порта протокола.

Ориентированная на соединение программа-клиент вызывает функцию connect, чтобы настроить сокет на сетевое соединение. Функция connect размещает информацию о локальной и удаленной конечных точках соединения в структуре данных сокета. Функция connect требует, чтобы были указаны: дескриптор сокета (указывающий на информацию об удаленном компьютере) и длина структуры адресных данных сокета. Следующий оператор — пример обращения к функции connect:

result = connect(socket_handle, remote_socket_address, address_length);

Первый параметр функции connect, дескриптор сокета, получен ранее от функции socket. Функция socket всегда вызывается до того, как устанавливается соединение. Дескриптор сокета указывает программному обеспечению, какая именно структура данных в таблице дескрипторов имеется в виду. Дескриптор также сообщает о том, куда нужно записать информацию об адресах удаленного участника соединения.

Второй параметр функции connect — адрес удаленного сокета, является указателем на структуру данных адреса сокета специального вида. Информация об адресе, хранящаяся в структуре, зависит от конкретной сети, то есть от семейства протоколов, которое мы используем. Во второй части книги вы узнаете больше об этой структуре. На данный момент нам важно знать, что структура данных сокета содержит семейство адресов, порт протокола и адрес сетевого компьютера. Функция connect записывает эту информацию в таблицу дескрииторов сокетов, на которую указывает соответствующий дескриптор сокета (первый параметр функции connect).

До того как вызвать функцию connect, информацию об адресах удаленного компьютера нужно занести в структуру данных сокета. Другими словами, функция connect должна знать сетевой адрес и номер порта удаленного компьютера. Местный IP-адрес, однако, можно не указывать. Реализация сокетов вашей операционной системы самостоятельно разместит адрес вашего компьютера и номер локального порта протокола. В добавок ко всему интерфейс сокетов проследит за тем, получило ли приложение данные, доставленные в локальный порт протокола транспортным уровнем сети. Другими словами, интерфейс сокетов сам выбирает локальный порт протокола для приложения и уведомляет его о получении данных — прикладная программа может не заботиться о том, какой именно порт она использует.

Третий параметр функции connect, длина адреса, сообщает интерфейсу длину структуры данных адресов удаленного сокета (второй параметр), измеренную в байтах. Содержимое (и длина) этой структуры зависит от конкретной сети. Зная длину структуры, интерфейс сокетов представляет, сколько памяти отведено для хранения этой структуры. Когда реализация сокетов выполняет функцию connect, она извлекает количество байтов, указанное третьим параметром из буфера данных, на который указывает параметр “адрес удаленного сокета”.

Указание локального адреса (порта протокола)

Функция connect устанавливает прямое соединение с удаленным сетевым компьютером. Это необходимо, только если используется ориентированный на соединение протокол. Если протокол не ориентирован на соединение, он никогда не устанавливает его напрямую. Не ориентированный на соединение протокол передает данные в датаграммах — он никогда не передает их потоком байтов. Точно так же программа-сервер никогда не начинает соединение первой. Вы можете создать программу-сервер, работающую по ориентированному на соединение протоколу, однако и в этом случае она будет пассивно прослушивать порт протокола, ожидая появления запроса от клиента. Другими словами, прямое соединение инициируется клиентом, а не сервером.

Ключевое сходство между любой не ориентированной на соединение программой и программой-сервером, ориентированной на соединение, состоит в том, что обе они прослушивают порт протокола. Например, ориентированные и не ориентированные на соединение программы-серверы должны ждать появления запроса клиента на порту протокола. Точно так же, поскольку не ориентированные на соединение клиенты не устанавливают соединения напрямую с удаленным компьютером, они должны ожидать появления датаграммы-ответа на собственный запрос на порту протокола. Функция bind интерфейса сокетов позволяет программам связать локальный адрес (совокупность адресов локального компьютера и номера порта) с сокетом. Следующий оператор иллюстрирует вызов функции bind:

result = bind(socket_handle, local_socket_address, address_length) ;

Создавая программу-сервер, вы закладываете в нее способность ожидать появления запроса от клиента. Вы знаете, что транспортный уровень TCP/IP общается с приложениями (клиентами и серверами) через порт протокола.

Другими словами, чтобы принять запрос клиента, сервер должен ожидать, что транспортный уровень доставит его на определенный номер порта протокола. Программа-сервер должна использовать функцию bind, чтобы зарегистрировать собственный порт протокола в рамках интерфейса сокетов. Программа должна сообщить интерфейсу, на какой из портов нужно доставлять данные, предназначенные именно этому серверу. Реализация

сокетов, в свою очередь, сообщает транспортному уровню, что определенный порт протокола занят приложением и что он должен доставлять все данные, адресованные этому порту, интерфейсу сокетов.

Как было замечено, не .ориентированный на соединение клиент также должен прослушивать порт протокола. Программы, пользующиеся не ориентированными на соединение протоколами, не устанавливают прямого сетевого соединения. Из этого следует, что не ориентированный на соединение клиент также должен прослушивать порт протокола на предмет появления ответных дата-грамм.

Как и программы-серверы, клиенты, не ориентированные на соединение, используют функцию bind, чтобы зарегистрировать порт протокола в интерфейсе сокетов. Другими словами, не ориентированный на соединение клиент, так же, как и сервер, указывает интерфейсу сокетов тот порт, который он будет использовать для доставки данных. Реализация сокетов организует в свою очередь интерфейс между таким клиентом и программным модулем UDP транспортного уровня. В последующих разделах этой главы обсуждаются функции интерфейса сокетов (такие, как listen, accept, recvfrom и recv), используемые прикладными программами для получения данных из порта протокола. На настоящий момент вы должны понимать, что для того, чтобы сконфигурировать сокет на определенный порт протокола, необходимо вызвать функцию bind.

Передача данных через сокет

После того как сокет сконфигурирован, через него можно установить сетевое соединение. Процесс сетевого соединения подразумевает посылку и прием информации. Интерфейс сокетов включает несколько функций для выполнения этих обеих задач. В этом разделе описывается, как программа посылает данные через сокет. Интерфейс сокетов Беркли обеспечивает пять функций для передачи данных через сокет. Эти функции разделены на две группы. Трем из них требуется указывать адрес назначения в качестве аргумента, а двум остальным — нет. Основное различие между двумя группами состоит в их ориентированности на соединение.

Передача данных через соединенный сокет

Виртуальные цепи образуются ориентированными на соединение протоколами. При этом соединение между конечными точками сети выглядит, как выделенное двухточечное соединение. После того как программное обеспечение установило соединение, прикладная программа может обмениваться данными в простом последовательном потоке байтов.

Функции интерфейса сокетов, осуществляющие ориентированную на соединение передачу данных, не требуют от прикладных программ указывать адрес назначения в качестве аргумента. Вся информация об адресах назначения (сетевой адрес и номер порта) заносится в структуру дескриптора сокета на стадии его конфигурации. В течение сеанса связи через ориентированный на соединение сокет, все заботы, связанные с адресами удаленного сокета и управления интерфейсом с транспортным уровнем, берет на себя реализация сокетов. Функции send, write и writev предназначены только для соединенных сокетов — они не позволяют программе указать адрес назначения. Все три функции требуют, чтобы в качестве первого аргумента при вызове был задан дескриптор сокета. Функция writev, в отличие от write, не требует, чтобы данные занимали непрерывную область памяти. В качестве аргумента функции writev может передаваться массив адресов, по которым расположены данные. Следующий оператор демонстрирует типичный вызов функции write:

result = write(socket_handle, message_buffer, buffer_length);

Дескриптор сокета обозначает структуру в таблице дескрипторов, содержащую информацию о данном сокете. Второй параметр функции write, буфер сообщения, указывает на буфер, то есть область памяти, в которой расположены предназначенные для передачи данные. Прикладная программа должна предварительно отвести память для этого буфера, а затем заполнить его данными. Третий параметр вызова функции обозначает длину буфера, то есть количество данных для передачи. Функция writev вызывается так, как показано ниже:

result = writev(socket_handle, io_vector, vector_length) ;

Так же, как и в случае write, функция writev требует, чтобы первым параметром указывался дескриптор сокета. Ее второй параметр, вектор ввода-вывода, указывает на массив указателей. Предположим, что данные для передачи располагаются в различных областях памяти. В этом случае каждый член массива указателей представляет собой указатель на одну из областей памяти, содержащей данные для передачи. Когда функция writev передает данные, она находит их по указанным прикладной программой в массиве указателей адресам. Данные высылаются в том порядке, в каком их адреса указаны в массиве указателей. Третий параметр функции writev определяет количество указателей в массиве указателей, заданном вектором ввода-вывода.

Как было отмечено, второй параметр функции writev является вектором ввода-вывода, то есть задает адрес массива, состоящего из последовательности указателей на блоки данных для передачи. С каждым указателем в последовательности интерфейс сокетов связывает определенную длину блока данных, то есть сколько байтов данных присутствует по каждому адресу в последовательности.

Поскольку функция write использует непрерывную область памяти в качестве буфера данных, то и выполняется она быстрее, чем writev. Однако в ситуации, когда структура передаваемых данных сложна либо нет возможности получить у системы достаточно большой непрерывный блок памяти, выручает функция writev. функция send — последняя из рассматриваемых функций, позволяющих отправлять данные через соединенный сокет. Следующий оператор является образцом вызова функции send:

result = send(socket_handle, message_buffer, buffer_length, special_flags) ;

Основное преимущество send состоит в том, что приложение может задать некоторые флаги для управления передачей данных. Вы знаете, что, например, TCP/IP имеет режим передачи данных вне основной полосы пропускания (данных для неотложной обработки). Эти данные имеют приоритет выше, чем у остальных. Один из возможных флагов в вызове send может сообщить, что приложение передает данные, требующие немедленной обработки.

Несмотря на то, что функция send позволяет передавать данные для неотложной обработки, вы не должны использовать этот режим, если четко не представляете, как они будут обработаны приемником. Теоретически, передача неотложных данных — мощное средство, но его конкретные реализации сложны и часто не совместимы друг с другом.

Три вышеописанные функции (write, writev и send) возвращают целое число в качестве результата. Если не произошла ошибка, результат будет равен количеству переданных байтов. В случае ошибки, возвращаемое значение результата будет равно -1. Сокеты Беркли устанавливают стандартную переменную errno языка C для того, чтобы сообщить дополнительную информацию об ошибке.

Передача данных через не соединенный сокет

Чтобы послать данные через соединенный сокет, то есть сокет протокола, ориентированного на соединение, прикладная программа может использовать одну из трех описанных в предыдущем разделе функций. Однако ни одна из них не позволяет указывать адрес получателя данных. Для того чтобы послать данные через не соединенный сокет, требуется вызвать одну из двух следующих функций: sendto или sendmsg. Они обеспечиваются интерфейсом сокетов именно для этих целей. Функция sendto требует шесть параметров в качестве аргументов. Первые четыре те же, что и в функции send. Пятый параметр, структура адреса сокета, определяет адрес назначения. Шестой параметр, длина структуры адреса сокета, - размер этой структуры в байтах. Следующий оператор демонстрирует вызов функции sendto:

result = sendto(socket_handle, message_buffer, buffer_length, special_flags, socket_address_structure, address_structure_length) ;

Функция sendmsg позволяет использовать гибкую структуру данных вместо буфера, расположенного в непрерывной области памяти. Следующий оператор демонстрирует вызов sendmsg. В качестве аргументов указываются дескриптор сокета, указатель на структуру данных и дополнительные флаги:

result = sendmsg(socket_handle, message_structure, special_flags);

Структура сообщения позволяет программе гибко размещать длинные списки параметров сообщения в единой структуре данных. Функция sendmsg похожа на writev в том, что прикладная программа может разместить свои данные в нескольких раздельно расположенных блоках памяти. Другими словами, как и в функции writev, структура сообщения содержит указатель на массив адресов памяти.

Прием данных через сокет

В интерфейсе сокетов есть пять функций, предназначенных для приема информации. Они называются: read, readv, recv, recvfrom, recvmsg и соответствуют функциям, использующимся для передачи данных. Например, функции recv и send обладают одинаковым набором параметров. Функция recv принимает данные, а send — посылает. Точно так же одинаков набор параметров и у функций writev и readv. Функция writev передает данные, а readv — принимает.

И та и другая позволяют задать массив адресов памяти, где располагаются данные. Функции recvfrom и recvmsg соответствуют функциям sendto и sendmsg. В табл. 3 приведен список всех соответствующих функций:

Несмотря на то, что интерфейс сокетов имеет соответствующие друг другу функции приема и передачи данных, никто не обязывает соблюдать это соответствие. Предположим, удаленный сетевой компьютер желает передать данные вашему приложению. Чтобы передать программе пришедшие из сокета данные, вовсе не обязательно вызывать строго соответствующую функцию. Так или иначе, данные в сокете все равно представлены единым потоком байтов. Поэтому считать их можно любой из имеющихся функций: recv, read или readv.

Интерфейс сокетов позволяет использовать наиболее удобную в данный момент функцию. Например, если программа не хочет запрашивать у системы большие непрерывные блоки памяти, она может пользоваться функцией readv. В любом случае необходимо учитывать тот факт, что исходный текст программы легче всего читается, если в нем для решения одинаковых задач всегда вызываются однотипные функции.

Сокеты и серверы

Обыкновенная, ориентированная на соединение программа-сервер вызывает функции listen и accept. Первая переводит сервер в режим пассивного ожидания запроса. Функция accept, наоборот, заставляет сокет программы установить соединение. В следующих абзацах описано, как и почему серверы используют эти две функции специального назначения.

Большинство операционных систем являются многопоточными. *Поток* (thread – потоки также называются легковесными процессами (lightweight processes) и иногда нитями (threads).) — это цепочка инструкций, из которых составлена программа. Каждое отдельное приложение в системе представлено набором потоков, которые можно считать процессами. Многопоточная программа (процесс) может состоять из множества одновременно и независимо исполняющихся потоков. Если многопоточная программа исполняется на многопроцессорном компьютере, каждый поток может исполняться на собственном процессоре. Очевидно, что многопоточная

программа в этом случае исполняется значительно быстрее по сравнению с программой из одного потока.

Вдобавок ко всему, многопоточная программа на компьютере с одним процессором оказывается быстрее программы из одного потока, который исполняется в одном процессе. Переключение процессов (контекста) на многозадачных компьютерах занимает больше времени, нежели переключение потоков. Конструкция сервера параллельной обработки предполагает, что для каждого нового запроса от клиента создается новая копия процесса-сервера. Если сервер параллельной обработки многопоточный, его производительность значительно увеличивается. Вместо создания нового процесса или их переключения, многопоточный сервер параллельной обработки просто образует новый поток, выполняющийся гораздо быстрее. Если сетевой компьютер оборудован несколькими процессорами, сервер будет работать еще быстрее, поскольку каждый поток сможет выполняться на отдельном процессоре.

Функция listen

Серверы бывают последовательной и параллельной обработки. Процесс-сервер, обрабатывающий каждый запрос клиента индивидуально, является последовательным. Последовательный сервер обрабатывает запросы поочередно, выстраивая их в очередь, если необходимо. Чтобы быть эффективным, такой сервер должен затрачивать ограниченное и заранее предсказуемое время на обработку поступающих запросов. Если сервер должен одновременно обработать несколько поступивших запросов, когда заранее неизвестно, сколько времени будет затрачено на обработку каждого, он конструируется параллельным. Параллельный сервер создает отдельный процесс (или поток, если это позволяет операционная система) для обработки каждого запроса. Другими словами, он обрабатывает запросы параллельно.

Теперь рассмотрим, что происходит, когда появляется очередной запрос, а сервер еще не закончил обработку предыдущего. То есть когда программа-клиент пытается послать еще один запрос, в то время как последовательный сервер еще не закончил обработку предыдущего, или когда клиент посылает запрос до того момента, как параллельный сервер закончил создание нового процесса — обработчика предыдущего запроса.

В этом случае сервер может отвергнуть или игнорировать поступивший запрос. Для того чтобы обработка была эффективнее, существует функция listen. Она, действуя как можно быстрее, располагает все поступившие запросы в очередь. Функция listen не только переводит сокет в пассивный режим ожидания, но и подготавливает его к обработке множества одновременно поступающих запросов. Другими словами, в системе организуется очередь поступивших запросов, и все запросы, ожидающие обработки сервером, помещаются в нее, пока освободившийся сервер не выберет его.

При вызове функции listen указываются два параметра: дескриптор сокета и длина очереди. Длина очереди обозначает максимальное количество запросов, которое может поместиться в ней. Следующий оператор — образец вызова функции listen:

result = listen(socket_handle, queue_length);

В настоящее время максимальная длина очереди равна пяти. Если вы попытаетесь указать большее число, то получите сообщение об ошибке. Если очередь при поступлении нового запроса окажется переполненной, сокет отвергнет соединение и программа-клиент получит сообщение об ошибке.

В случае последовательного сервера, задавать длину очереди, равную одному или двум, тоже полезно. Это позволит серверу, если он не справился с обработкой за минимальное назначенное время, все-таки не отвергнуть новый запрос, а выбрать его из входной очереди.

Функция accept

Как уже отмечалось, функция bind привязывает сетевой адрес и номер локального порта протокола к определенному сокету. Программы-клиенты устанавливают соединение, привязывая сокет к удаленному порту протокола, расположенному на удаленном компьютере. Клиент знает, какой именно сервер ему нужен, поэтому располагает всей требуемой информацией.

С другой стороны, сервер не имеет представления о том, от какого клиента может поступить запрос. Поэтому, вызвав bind, он не может определить заранее ни его адреса, ни номера порта. Вместо этого сервер вызывает bind, задав в качестве аргумента символ, совпадающий с любым адресом (wildcard). Константа, определяющая любой адрес, называется INADDR_ANY. Другими словами, такой сервер принимает запросы от любого сетевого компьютера, то есть от любой программы-клиента.

Для того чтобы указать “любой” адрес, используется символьная константа INADDR_ANY, описанная в файле-заголовке winsock.h.

Как следует из названия, функция accept позволяет серверу принять запрос на соединение, поступивший от клиента. После того как установлена входная очередь, программа-сервер вызывает функцию accept и переходит в режим ожидания (паузы), ожидая запросов. Для того чтобы понять смысл всех операций сервера, мы должны подробно рассмотреть, как функционирует accept.

При вызове accept требуется указывать три параметра: дескриптор сокета, его адрес и длину адреса. Дескриптор сокета описывает сокет, который будет прослушиваться сервером. В момент появления запроса реализация сокетов заполняет структуру адреса (на которую указывает второй параметр) адресом клиента, от которого поступил запрос.

Реализация сокетов заполняет также третий параметр, размещая в нем длину адреса. Следующий оператор

демонстрирует, как можно вызывать функцию accept:

result = accept(socket_handle, socket_address, address_length);

После того как реализация сокетов поместит информацию об адресе клиента в область памяти, заданную параметром функции, она выполнит операции, необходимые для того, чтобы сервер заработал. Первым делом, получив запрос соединения на сокет, обслуживаемый функцией accept, реализация образует новый сокет. Новый сокет связывается с адресом процесса-передатчика запроса.

Коротко весь процесс можно описать следующим образом: когда на сокете, контролируемом функцией accept, появляется очередной запрос клиента, программное обеспечение-реализация сокетов автоматически создает новый сокет и немедленно соединяет его с процессом-клиентом. Сокет, на который поступил запрос, освобождается и продолжает работу в режиме ожидания запросов от любого сетевого компьютера.

Процесс-сервер

Как только на сокете, обслуживаемом функцией accept, появляется запрос, функция возвращает серверу дескриптор только что созданного нового сокета. Что сервер будет делать с этим дескриптором, зависит от реализации сервера. Сервер может обрабатывать запросы параллельно или последовательно. Предположим, что вы создали последовательный сервер. Следовательно, он будет последовательно обрабатывать, а затем закрывать все переданные ему функцией accept дескрипторы сокетов. По окончании обработки конкретного запроса последовательный сервер вновь вызывает accept, а она возвращает ему дескриптор сокета для следующего запроса, если он имеется, и т. д. Если до этого вызывалась функция listen, запрос может быть выбран из входной очереди, если нет — сервер прослушивает сетевые запросы напрямую через сокет.

Протокол SMTP.

Электронная почта (e-mail) является самым широко используемым приложением для большинства сетей. Действительно, половина всех устанавливаемых TCP/IP-соединений предназначена для обмена электронной почтой.

Концепции электронной почты можно применить везде, где требуется разработать простую, но эффективную сетевую службу Интернет. Можно реализовать дополнительный сервис на базе почтового, что не потребует разработки новых протоколов — достаточно будет уже имеющегося. Расширения службы электронной почты дают возможность обмениваться не только текстовыми сообщениями, но и передавать двоичные файлы, например изображения, звуки или исполняемые файлы.

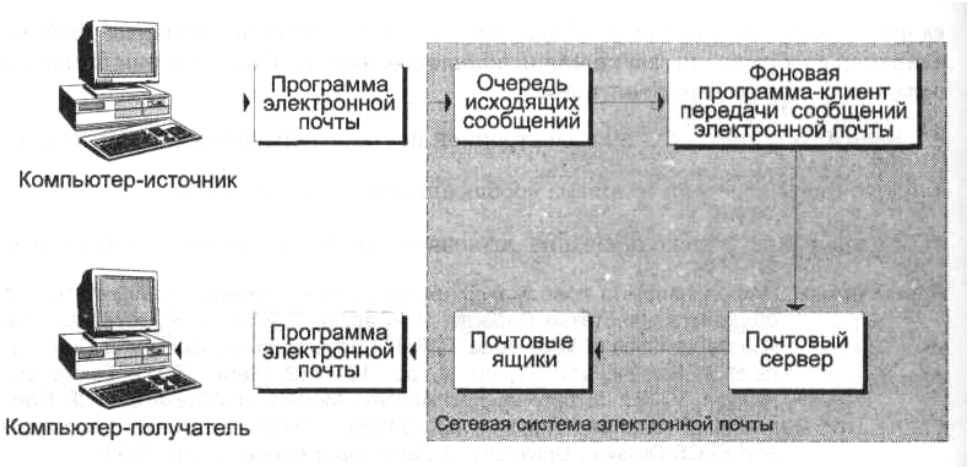

Рис. 1. Концептуальные составляющие электронной почты.

На рисунке 1 изображены основные (концептуальные) составляющие системы электронной почты Интернет. Каждому сообщению соответствует источник и получатель. И источник и получатель обладают каким-либо пользовательским интерфейсом к сетевой почтовой системе.

В общих чертах, почтовая система состоит из очереди исходящих сообщений, процесса-клиента, процесса-сервера и почтовых ящиков пользователей, хранящих доставленные сообщения. Несмотря на то, что в почтовую систему встраивается пользовательский интерфейс, он не обязан там присутствовать. То есть он может представлять собой отдельную программу, устроенную по принципу клиент-сервер и взаимодействующую с самой почтовой системой. *Почтовый ящик* (mailbox) носит соответствующее своему владельцу название. Им является либо адрес в формате «имя_пользователя@имя компьютера.имя_домена» либо файл-контейнер для промежуточной доставки. *Файл-контейнер* аналогичен местному почтовому отделению. Файл-контейнер хранит сообщение до тех пор, пока пользователь не заберет его оттуда.

Компоненты электронной почты Интернет

На рисунке 2 приведены настоящие компоненты системы электронной почты, в отличие от концептуальных на рисунке 1. Обратите внимание на термины *«агент пользователя»* (user agent, UA) и *«агент передачи почты»* (message transfer agent, MTA). Как видим, агент пользователя заменяет почтовую программу, а агент передачи почты заменяет процесс-клиент и процесс-сервер.

Термин *«агент»* довольно часто встречается в документации Интернет. «Агент» — это программа специального назначения, выполняющая действия для пользователя или другой программы. В большинстве случаев почтовая программа называется агентом пользователя (UA). Точно так же агент передачи почты (MTA) представляет собой клиент или сервер, выполняющий задачи по доставке или получению почты на сетевом компьютере.

Вы, как пользователь, взаимодействуете с агентом пользователя. Он, в свою очередь, взаимодействует с файлом-контейнером или агентом передачи сообщений за вас. В то же время, MTA ведет себя как представитель своего компьютера в сети. Агент пользователя защищает вас от необходимости общаться с различными почтовыми хостами, а MTA защищает компьютер от необходимости общаться с различными агентами пользователя или несколькими агентами передачи почты одновременно.

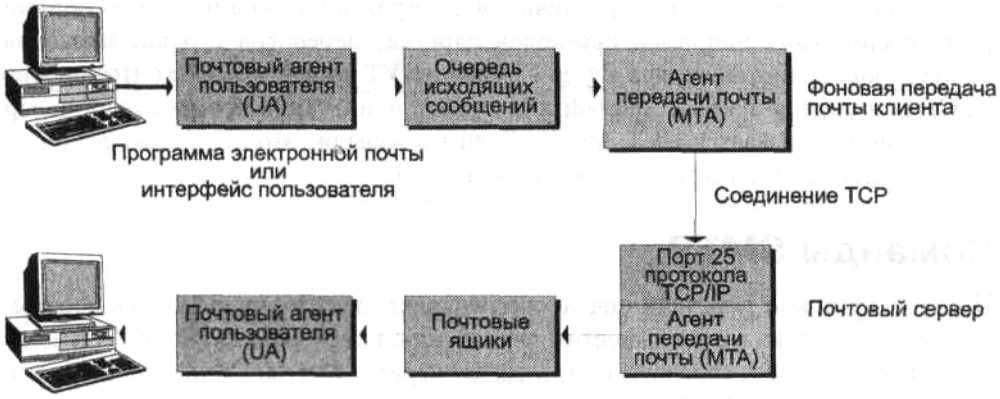

Рисунок 2. Настоящие компоненты электронной почты.

В принципе, пользовательский агент отделен от агента передачи почты. Конечно, их можно объединить в одной программе, но все равно это будут отдельные модули. Задача агента пользователя — создать для него удобный и дружелюбный интерфейс к почтовой системе.

Система электронной почты представлена агентами передачи почты, MTA. До того как обсудить задачи пользовательского агента, необходимо узнать немного больше о том, что же такое MTA. MTA умеют устанавливать TCP-соединение для связи с другими MTA. Протоколом этого соединения, как правило, является простой протокол передачи почты (SMTP). Этот протокол полностью описан в RFC 821, который так и называется *«Простой протокол передачи почты»* (Simple Mail Transfer Protocol, Postel, 1982).

Простой протокол передачи почты (SMTP)

Агент передачи почты — основной компонент системы передачи почты Интернет. Как уже говорилось, MTA как бы представляет данный сетевой компьютер для сетевой системы электронной почты. Пользователи редко имеют дело с MTA, поскольку он не вполне «дружелюбен», однако без него не обходится ни одна почтовая система. После того как UA пошлет сообщение в выходную очередь, за дело принимается MTA. Он извлекает сообщение и посылает его другому MTA. Этот процесс продолжается до тех пор, пока сообщение не достигнет компьютера-получателя. Для передачи сообщений по TCP-соединению большинство MTA пользуются протоколом SMTP. Сообщения форматированы по правилам виртуального сетевого терминала (NVT), то есть в NVT ASCII. Символ в NVT состоит из семи битов набора ASCII и является буквой, цифрой или знаком пунктуации. Семибитный набор ASCII часто называется NVT ASCII.

Команды SMTP

Простой протокол передачи почты обеспечивает двухсторонний обмен сообщениями между локальным клиентом и удаленным сервером MTA. MTA-клиент шлет команды MTA-серверу, а он, в свою очередь, отвечает клиенту. Другими словами, протокол SMTP требует получать ответы (они описаны в этой главе) от приемника команд SMTP. Обмен командами и ответами на них называется *почтовой транзакцией* (mail transaction). Данные передаются в формате NVT ASCII. Кроме того, команды тоже передаются в формате NVT ASCII. Команды передаются в форме ключевых слов, а не специальных символов, и указывают на необходимость совершить ту или иную операцию. В табл. 1 приведен список ключевых слов (команд), определенный в спецификации SMTP - RFC 821.

Таблица 1. Команды простого протокола передачи почты (SMTP)

Команда	**Обязательна**	**Описание**
HELO	**X**	Идентифицирует модуль-передатчик для модуля-приемника (hello).
MAIL	**X**	Начинает почтовую транзакцию, которая завершается передачей данных в один или несколько почтовых ящиков (mail).
RCPT	**X**	Идентифицирует получателя почтового сообщения (recipient).
DATA		Строки, следующие за этой командой, рассматриваются получателем как данные почтового сообщения. В случае SMTP, почтовое сообщение заканчивается комбинацией символов: CRLF-точка-CRLF.
RSET		Прерывает текущую почтовую транзакцию (reset).
NOOP		Требует от получателя не предпринимать никаких действий, а только выдать ответ OK. Используется главным образом для тестирования. (No operation.)
QUIT		Требует выдать ответ OK и закрыть текущее соединение.
VRFY		Требует от приемника подтвердить, что ее аргумент является действительным именем пользователя. (См. примечание.)
SEND		Начинает почтовую транзакцию, доставляющую данные на один или несколько терминалов (а не в почтовый ящик).

SOML		Начинает транзакцию MAIL или SEND, доставляющую данные на один или несколько терминалов или в почтовые ящики.
SAML		Начинает транзакцию MAIL и SEND, доставляющие данные на один или несколько терминалов и в почтовые ящики.
EXPN		Команда SMTP-приемнику подтвердить, действительно ли аргумент является адресом почтовой рассылки и если да, вернуть адрес получателя сообщения (expand).
HELP		Команда SMTP-приемнику вернуть сообщение-справку о его командах.
TURN		Команда SMTP-приемнику либо сказать ОК и поменяться ролями, то есть стать STMP-передатчиком, либо послать сообщение-отказ и остаться в роли SMTP-приемника.

Примечание: В RFC 821 сказано, что команда VRFY не является обязательной для минимального набора команд SMTP. Однако в RFC 1123 «Требования для сетевых компьютеров Интернет — приложения и обеспечение работы» (Requirements for Internet Hosts — Application and **Support,** Braden, 1989), команда VRFY фигурирует в списке обязательных для Интернет команд реализации **SMTP.**

В соответствии со спецификацией команды, помеченные крестиком (X) в табл. 1, обязаны присутствовать в любой реализации SMTP. Остальные команды SMTP могут быть реализованы дополнительно. Каждая SMTP-команда должна заканчиваться либо пробелом (если у нее есть аргумент), либо комбинацией CRLF. В описании команд употреблялось слово *«данные»,* а не «сообщение

Коды ответов SMTP

В спецификации SMTP требуется, чтобы сервер отвечал на каждую команду SMTP-клиента. MTA-сервер отвечает трехзначной комбинацией цифр, называемой кодом ответа. Вместе с кодом ответа, как правило, передается одна или несколько строк текстовой информации.

Примечание: Несколько строк текста, как правило, сопровождают только команды EXPN и HELP. В спецификации SMTP, однако, ответ на любую команду может состоять из нескольких строк текста.

Каждая цифра в коде ответа имеет определенный смысл. Первая цифра означает, было ли выполнение команды успешно (2), неуспешно (5) или еще не закончилось (3). Как указано в приложении E документа RFC 821, простой SMTP-клиент может анализировать только первую цифру в ответе сервера, и на основании ее продолжать свои действия. Вторая и третья цифры кода ответа разъясняют значение первой. То, как коды составлены в самом SMTP — превосходный образец грамотного подхода к делу.

Что означает первая цифра в коде ответа SMTP?

В спецификации SMTP для первой цифры кода ответа отведено пять возможных значений. Цифра 1 означает, что сервер MTA принял команду, от клиента требуется дополнительное подтверждение. Клиент обязан послать дополнительную информацию о том, продолжать или прервать выполнение запрошенной команды.

Коды ответа, начинающиеся с цифры 2, означают, что сервер MTA успешно завершил выполнение команды и ожидает появления новой. Код ответа, начинающийся на 3, означает, что команда начала выполняться, но серверу необходима дополнительная информация для ее завершения. Пример такого кода — 354. В ответ на него клиент MTA должен приступить к передаче почтового сообщения. Код, начинающийся с цифры 4, означает, что сервер не принял команду и она, соответственно, не выполнена. Однако во всех ответах серии 400 предполагается, что ошибка временная и клиент может попытаться ее исправить. Коды ответа серии 500 также сообщают, что команда не выполнена. Кроме того, клиент не должен пытаться повторить ту же команду еще раз по крайней мере в составе той же последовательности.

Вторая цифра кода ответа SMTP

Вторая цифра кода ответа обозначает категорию ошибки. Цифра 0, например, обозначает синтаксическую ошибку. Команда может быть слишком длинной, иметь неправильный аргумент или, наконец, отсутствовать в списке команд сервера.

Вторая цифра — двойка, и все коды имеют дело с передачей данных или с коммуникационным каналом. Коды ответов, у которых вторая цифра равна пяти (250, 450 и 550), связаны непосредственно с почтовой системой. В настоящее время в SMTP не определены значения кодов, вторая цифра которых равна трем или четырем.

Третья цифра кода ответа SMTP

В спецификации SMTP указано, что каждая отдельная строка сообщения должна иметь собственную третью цифру в коде ответа.. Она обозначает номер сообщения в данной серии. Спецификация SMTP рекомендует, но не обязывает использовать строго заданные текстовые строки в ответах MTA-сервера.

Протокол Post Office Protocol (POP)

Протокол доставки почты пользователю из почтового ящика называется Post Office Protocol (POP). Команды POP практически идентичны командам SMTP, отличаясь в некоторых деталях. На рис. 3 изображена модель клиент-сервер по протоколу POP. Сервер POP находится между агентом пользователя и почтовыми ящиками.

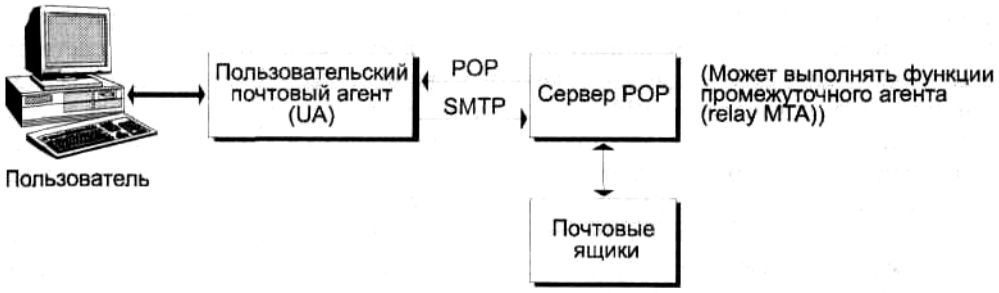

Рисунок 3. Конфигурация модели клиент-сервер по протоколу POP.

В настоящее время существуют две версии протокола POP — POP2 и POP3. Обе версии обладают примерно одинаковыми возможностями, однако несовместимы друг с другом. Дело в том, что у POP2 и POP3 разные номера портов протокола. Между ними отсутствует связь, аналогичная связи между SMTP и ESMTP. Протокол POP3 не является расширением или модификацией POP2 — это совершенно другой протокол. POP2 определен в документе RFC 937 (Post Office Protocol-Version 2, Butler, et al, 1985), а POP3 - в RFC 1225 (Post Office Protocol-Version 3, Rose, 1991).

Протокол POP3

Конструкция протокола POP3 обеспечивает возможность пользователю войти в систему и изъять накопившуюся почту, вместо того чтобы предварительно входить в сеть. Пользователь получает доступ к POP-серверу из любой системы в Интернет. При этом он должен запустить специальный почтовый агент (UA), понимающий протокол POP3. Во главе модели POP находится отдельный персональный компьютер, работающий исключительно в качестве клиента почтовой системы. В соответствии с этой моделью персональный компьютер не занимается ни доставкой ни авторизацией сообщений для других. Также сообщения доставляются клиенту по протоколу POP, а посылаются по-прежнему при помощи SMTP. То есть на компьютере пользователя существуют два отдельных агента-интерфейса к почтовой системе — доставки (POP) и отправки (SMTP). Разработчики протокола POP3 называет такую ситуацию *«раздельные агенты»* (split UA). Концепция раздельных агентов кратко обсуждается в спецификации POP3.

В протоколе POP3 оговорены три стадии процесса получения почты: авторизация, транзакция и обновление. После того как сервер и клиент POP3 установили соединение, начинается стадия авторизации. На стадии авторизации клиент идентифицирует себя для сервера. Если авторизация прошла успешно, сервер открывает почтовый ящик клиента и начинается стадия транзакции. В ней клиент либо запрашивает у сервера информацию (например, список почтовых сообщений), либо просит его совершить определенное действие (например, выдать почтовое сообщение). Наконец, на стадии обновления сеанс связи заканчивается. В табл. 2 перечислены команды протокола POP3, обязательные для работающей в Интернет реализации минимальной конфигурации.

Таблица 2. Команды протокола POP версии 3 (для минимальной конфигурации)

Команда	**Описание**
USER	Идентифицирует пользователя с указанным именем
PASS	Указывает пароль для пары клиент-сервер
QUIT	Закрывает TCP-соединение
STAT	Сервер возвращает количество сообщений в почтовом ящике плюс размер почтового ящика
LIST	Сервер возвращает идентификаторы сообщений вместе с размерами сообщений (параметром команды может быть идентификатор сообщения)
RETR	Извлекает сообщение из почтового ящика (требуется указывать аргумент-идентификатор сообщения)
DELE	Отмечает сообщение для удаления (требуется указывать аргумент-идентификатор сообщения)
NOOP	Сервер возвращает положительный ответ, но не совершает никаких действий
LAST	Сервер возвращает наибольший номер сообщения из тех, к которым ранее уже обращались
RSET	Отменяет удаление сообщения, отмеченного ранее командой DELE

В протоколе POP3 определено несколько команд, но на них дается только два ответа: +OK (позитивный, аналогичен сообщению-подтверждению ACK) и -ERR (негативный, аналогичен сообщению «не подтверждено» NAK). Оба ответа подтверждают, что обращение к серверу произошло и что он вообще отвечает на команды. Как правило, за каждым ответом следует его содержательное словесное описание. В RFC 1225 есть образцы нескольких типичных сеансов POP3. Сейчас мы рассмотрим несколько из них, что даст возможность уловить последовательность команд в обмене между сервером и клиентом.

WWW и протокол HTTP.

Протокол передачи гипертекста

Подобно всему в Интернет действия WWW зависят от протокола передачи гипертекста (HTTP). Так же как в FTP, POP и SMTP, в HTTP задан набор команд, передающийся посредством строк текста в формате ASCII. Транзакция HTTP еще более проста, чем транзакции FTP. Транзакция HTTP состоит из четырех частей: установление соединения, запрос, ответ и завершение. Программа-клиент HTTP устанавливает TCP-соединение с официальным портом HTTP (80) на удаленном компьютере. Затем клиент посылает запрос к серверу HTTP. После того как сервер HTTP высылает ответ, клиент или сервер закрывают соединение. Каждая транзакция HTTP подчиняется вышеописанной схеме. Также в большинстве случаев ответ сервера состоит из передачи запрошенного файла потоком байтов в локальный порт протокола клиента. HTTP не требует никаких специальных действий по завершению соединения.. Как клиент, так и сервер (или оба сразу) имеют право закрыть TCP-соединение.

Запросы клиента HTTP

Web обеспечивает пользователей сети свободным доступом к огромному количеству файлов — ресурсам Интернет. Как уже говорилось, после установления соединения клиент HTTP может послать HTTP-запрос. Запрос клиента HTTP состоит обычно из просьбы к HTTP-серверу передать файл (гипертекстовый документ, изображение, звуковой файл, мультипликацию или видео) от сервера к клиенту.

Для получения файла требуется, чтобы программа-клиент WWW передала имя определенного файла, его местоположение в Интернет (адрес хоста) и метод передачи (обычно протокол типа HTTP или FTP). Комбинация этих элементов формирует нечто, названное универсальным идентификатором ресурса (Universal Resource Identifier, URI).

Указатели ресурсов: URI и URL

Чтобы получить файл из Интернет, браузер (browser, программа для просмотра Web) должен знать, где находится файл и как общаться с компьютером, на котором этот файл находится. На сегодняшний день браузеры WWW используют протокол HTTP и несколько других общих для Интернет протоколов типа FTP, GOPHER, ARCHIE, VERONICA и WAIS. В состав URI входит информация, требующаяся браузеру WWW, чтобы использовать любой из этих протоколов. Различие между URI и URL достаточно сложно уловить. Вообще, вы можете рассматривать некоторый URI как идентификатор определенного объекта типа файла изображения или HTML-файла. URI является как бы обобщенным названием любого объекта в Интернет. Однако если вы хотите использовать существующие протоколы, чтобы получить объект из Интернет, вы должны знать IP-адрес компьютера, хранящего этот объект. URL является URI, который содержит информацию о местоположении (адрес), закодированную в URI.

Синтаксис URL

Как уже говорилось, URL содержит информацию об адресе или местоположении объекта. Цель, преследуемая URL, состоит в том, чтобы дать возможность другим программам найти объект в Интернет.

Практически, URL можно рассматривать, как специальный тип сетевого адреса, который, однако, не просто идентифицирует сетевой компьютер, но и идентифицирует определенный объект на нем. Кроме того, в URL задается другая важная характеристика, отличающая его от других типов сетевых адресов, — метод доступа к сетевому объекту.

В RFC 1738 методы доступа называются *схемами.* Другими словами, схема URL описывает, каким образом программа может получить определенный сетевой объект. На сегодняшний день методы доступа соответствуют сетевым протоколам. В табл. 3 содержится список определенных в настоящее время схем доступа. Как видим, некоторые схемы доступа URL — не что иное, как протоколы, изученные вами в предыдущих главах этой книги.

Таблица 3. Определенные на настоящий момент схемы доступа URL

Схема доступа (элемент URL)	**Описание**
ftp	Протокол передачи файлов
http	Протокол передачи гипертекста
gopher	Протокол Gopher
mailto	Адрес электронной почты
news	Новости USENET
nntp	Новости USENET по протоколу NNTP
telnet	Сеанс telnet
wais	Сервер протокола wais
file	Имя файла в компьютере
prospero	С чужба каталогов prospero

Синтаксис URL прост, он состоит из двух частей, как показано ниже:

<схема>:<часть-относящаяся-к-схеме>

Первая часть URL содержит название схемы (обычно — название протокола), которая используется для доступа к

объекту. Вторая часть содержит специфическую для данной схемы информацию типа названия и местоположения объекта, а также информацию для определенного метода доступа. Программу клиент Telnet можно использовать для входа практически в любой компьютер Интернет. Однако в зависимости от того, в какую систему вы хотите войти, вам может понадобиться ввести имя пользователя и пароль. Схема URL Telnet обеспечивает необходимый для ввода пароля и имени синтаксис:

telnet://<пользователь>:<пароль>@<компьютер>:<порт>/

Так как в этой главе мы сконцентрируем внимание на WWW, в большинстве примеров URL будет использоваться протокол передачи гипертекста (HTTP). Как уже упоминалось, HTTP — специальный протокол, предназначенный для WWW. Подробная информация относительно других схем доступа URL находится в документе RFC 1738.

Определение схемы URL HTTP

Схема доступа URL HTTP просто определяет объекты Интернет, для доступа к которым может использоваться протокол передачи гипертекста (HTTP). Синтаксис схемы доступа URL HTTP (показанный ниже) похож на синтаксис схемы доступа Telnet:

http://<компьютер>:<порт>/<путь>?<информация_для_поиска>

Как видим, схема URL — HTTP, а в специфической для схемы части задаются: сетевой компьютер (имя или адрес), номер порта, путь и дополнительная информация для поиска. Если элемент «порт» в URL не указывается, будет использован порт 80 (официальный порт протокола HTTP). Элемент «информация_для_поиска» включен в синтаксис схем URL HTTP (в RFC 1738. Элемент «путь» задает каталог сетевого компьютера, в котором содержится интересующий нас объект (как правило, документ HTML), предназначенный для передачи по HTTP.

Методы HTTP

Как уже говорилось, запрос клиента HTTP (обычно запрос на передачу файла) — вторая часть транзакции HTTP. Запросы HTTP-клиента делятся на две основные категории: простой запрос и полный запрос. Запросы HTTP называются *методами.* Единственный метод простого запроса — метод GET. Как видим, при простом запросе программа просто передает URI и комбинацию CRLF после команды GET:

GET <uri> CRLF

Этот простой запрос заставит сервер HTTP найти и передать объект, который указан в URI. Объект может быть документом HTML, изображением, звуком, видео или файлом мультипликации. Так как клиент запрашивает совершенно определенный тип объекта, он должен знать, как обращаться с полученным объектом. Например, клиент, который запрашивает файл изображения, должен знать, как показать этот файл на дисплее компьютера. Для простого HTTP-запроса сервер HTTP является не более, чем файловым сервером.

Полный HTTP-запрос также начинается с метода HTTP. К сожалению, так как определения большинства предложенных методов HTTP для полных запросов несовершенны, эти методы в настоящее время не слишком полезны. После команды полного запроса следует URI и версия протокола HTTP. Полный запрос может также включать область запроса заголовка (header) (подобные областям заголовка MIME, связанным с SMTP). Однако каждый элемент передается отдельной строкой, оканчивающейся CRLF.

Код ответа сервера HTTP подобен кодам, используемым серверами SMTP, POP и FTP. Однако код ответа (или код состояния, как он называется в спецификации HTTP) — не первый элемент в строке ответа. Вместо этого в качестве первого элемента в строке ответа печатается номер версии:

<Версия_http> <код_состояния> <текст> <CRLF>

Протоколы передачи файлов

Хотя электронная почта, вероятно — самое широко используемое приложение Интернет, протокол передачи файлов (File Transfer Protocol, FTP) несет наибольшую нагрузку по передаче данных. В большинстве случаев до того, как получить доступ к файлам, нужно зарегистрироваться в системе. Чтобы облегчить процедуру получения доступа к файлам, многие системы разрешают анонимный доступ для пользователей по имени *anonymous.* С *анонимным* FTP пользователь может войти в систему FTP-сервера и передавать файлы не имея зарегистрированных прав доступа в системе. Другими словами, анонимный FTP позволяет людям свободно получать файлы данных из Интернет.

Протокол FTP управляет большинством операций по передаче файлов в Интернет. Для выполнения передачи файлов требуется установить два TCP-соединения, в отличие от других протоколов для которых необходимо только одно TCP-соединение. Одно TCP-соединение служит для передачи команд и ответов на них так же, как в POP3 или SMTP, а другое — для передачи собственно данных.

Основы FTP

Модель FTP

Протокол передачи файлов подобен SMTP и POP3 - в нем используются строки ASCII в качестве команд и кодов ответа. Однако в отличие от всех предыдущих протоколов FTP использует два TCP-соединения для выполнения операций по передаче файлов. В FTP два TCP-соединения определяются как *управляющее соединение* и *соединение данных.* Управляющее соединение подобно другим TCP-соединениям. Другими словами, управляющее соединение -типичное соединение клиент-сервер. Сервер FTP обеспечивает пассивное открытие на официальном порту (порт протокола 21) и ждет запроса на установление соединения от клиента. Клиент FTP, в свою очередь, входит в контакт с FTP-сервером на официальном порту протокола и устанавливает с ним TCP-соединение. Управляющее соединение остается активным на протяжении всего FTP-сеанса. Клиент и сервер обмениваются строками команд NVT ASCII и кодами ответа через управляющее соединение. FTP создает отдельное соединение данных для каждой операции по передаче файла (а также в некоторых других случаях). На рис. 4 изображена типичная конфигурация для операций по передаче файлов.

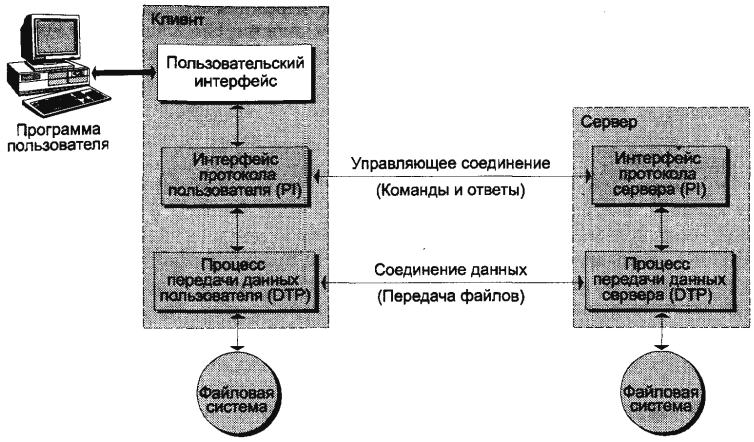

Рис. 4. Типичная конфигурация для операций по передаче файлов

Основа операции - интерпретаторы протокола (PI) и процессы передачи данных (DTP). Как видим, клиент и сервер имеют свой собственный интерпретатор протокола и процесс передачи данных. Процессы передачи данных устанавливают и управляют соединением данных. Интерпретаторы протокола интерпретируют FTP-команды и общаются через управляющее соединение которое устанавливается в начале FTP-сеанса интерфейсом протокола пользователя.

Интерфейс пользователя ограждает пользователя от непосредственного общения с командами и ответами FTP. Программы Ftp вообще бывают двух видов: полноэкранные (ориентированные на работу с меню) или строчные (ориентированные на работу с командной строкой). Большинство UNIX-программ Ftp — строчные.

Управление данными

С самого начала проектировщики протокола FTP разрабатывали его для работы с различными компьютерами, использующими различные операционные системы, структуры файлов и наборы символов. В результате FTP требует, чтобы пользователи выбрали необходимые пункты из широкого разнообразия опций для операций по передаче файла. Опции FTP можно разбить на четыре категории: тип файла, формат файла, структура файла и способы передачи. Следующие части описывают каждую из этих опций.

Что означает тип файла в FTP?

FTP может управлять четырьмя различными *типами файлов:* локальными, файлами изображений (или двоичными), EBCDIC и ASCII. Локальный тип файла предназначен для передачи файла между хостами, которые используют различные размеры байта. Как вы помните из обсуждения термина *октет* в третьей главе, много ранних разработок TCP/IP созданы на компьютерных системах, где байт не равнялся восьми битам. Локальный тип

файла — пережиток тех дней. Другими словами, локальный тип файла позволяет данным пользователя с компьютера, использующего 8 бит на байт, передаваться компьютеру, использующему другой размер байта (7 или 10 бит, например). Для системы, которая использует байты из 8 бит, локальный тип файла является идентичным типу файла “изображения” (двоичному файлу). Так как современные компьютеры используют байты, состоящие из 8 бит, локальный тип файла сегодня не слишком часто используется.

Файл типа “изображение” (или двоичный тип файла) передается как непрерывный поток данных. Другими словами, передача файла изображения не включает (и не идентифицирует) никаких границ во внутренней структуре файла данных (типа возврата каретки или перевода строки). Обычно пользователи FTP передают большинство файлов именно в этом режиме. Как вы узнали, большинство компьютеров для представления текстовых данных используют коды ASCII. Однако некоторые системы, типа универсальных ЭВМ фирмы IBM (мэйнфреймов) и мини-компьютеров, используют код EBCDIC. EBCDIC (произносится И-би-си-дик) расшифровывается как Extended Binary Coded Decimal Interchange Code. Хотя и EBCDIC и ASCII используют 8 битов для представления символа, их коды, соответствующие одинаковым символам, совершенно различны. Таким образом, компьютер, который понимает код EBCDIC, не будет понимать код ASCII (хотя существует большое количество программ, которые могут переводить из одной кодировки в другую).

Передача файла с типом EBCDIC в FTP — альтернативный метод передачи файла для двух компьютеров, которые используют кодировку EBCDIC. Другими словами, если хосты на каждом конце соединения FTP используют EBCDIC, они могут использовать тип файла EBCDIC, чтобы упростить передачу текстовых файлов. Передачи файла с типом ASCII — принимаемый по умолчанию тип для передачи файлов по FTP. Чтобы использовать передачу файла ASCII, посылающий хост должен преобразовать локальный текстовый файл в NVT ASCII (ASCII с 7 битами на символ). Хост-получатель должен перевести NVT ASCII к собственному соглашению для хранения текста.

Основная проблема с передачей файла ASCII — маркеры конца строки (end-of-line). Как вы узнали, маркер конца строки у различных компьютеров отличается от соглашения NVT ASCII, согласно которому конец строки отмечается парой символов CRLF (перевод строки, возврат каретки). Таким образом, хосты, использующие другие идентификаторы конца строки, должны просмотреть каждый байт поступающих данных, чтобы идентифицировать концы строк текста. Очевидно, такое требование добавляет работы по обработке данных программе-приемнику. Другими словами, приемник, который использует маркеры CRLF (то же соглашение, что используется при передаче файлов типа ASCII), может просто читать данные из очереди поступающих данных и записывать их в локальный файл. С другой стороны, приемник, который использует другое соглашение, должен исследовать каждую пару байтов в очереди поступающих данных, чтобы определить, когда программа должна заменить CRLF собственным маркером конца строки (иногда им бывает единственный символ перевода строки, LF).

Что такое формат файла в FTP?

Как сказано в предыдущей части, пользователь FTP может выбирать, как передавать файлы — с типом ASCII или с типом EBCDIC. Когда пользователь определяет ASCII или EBCDIC как тип файла для передачи, он должен также определить используемое *управление форматом*. FTP определяет три типа управления форматом: Nonprint, Telnet и FORTRAN. Для текстовых файлов используемое по умолчанию управление форматом — Nonprint, которое означает, что файл не содержит никакой информации о вертикальном формате типа вертикальной прокрутки страницы, которую принтер мог бы использовать для правильного расположения текста на бумаге. Управление форматом Telnet, напротив, использует управление вертикальным форматом для принтеров. Управление форматом Telnet — это символьные последовательности, которые сообщают принтеру, как печатать текст. FORTRAN (язык программирования) также использует специальные, вложенные в текст символы. Управление форматом FORTRAN означает, что первый символ каждой строки — символ управления FORTRAN, который, в свою очередь, определяет формат строки. Управление форматом Telnet и управление форматом FORTRAN — пережиток ранних дней Интернет. Сегодня большинство реализации FTP (особенно на UNIX-системах) ограничиваются управлением форматом Nonprint.

Режимы передачи FTP

Заключительный параметр для передачи файла по FTP, который должен быть определен пользователем, — *режим передачи.* Режим передачи определяет, как FTP передает файл через TCP-соединение. FTP определяет три режима передачи: блочный, со сжатием и потоковый. В блочном режиме файл передается как последовательность блоков, где каждый блок включает один или больше байтов заголовка. FTP посылает блоки данных файла в той же последовательности, в которой они находятся в файле. Заголовок в каждом блоке данных определяет размер блока данных (который может изменяться), а также дескрипторы, которые идентифицируют конец файла или конец записи (если таковые имеются). В режиме со сжатием простой алгоритм сжимает последовательные возникновения одного и того же байта — они кодируются специальным символом, сопровождаемым количеством одинаковых символов в последовательности.

В потоковом режиме FTP передает файл как непрерывный поток байтов. Если тип структуры FTP — структура с записями, FTP использует специальную двухбайтовую последовательность символов, чтобы отметить конец записи и конец файла. Если FTP использует файловую структуру, конец файла отмечается закрытием TCP-соединения. Другими словами, после того как передан последний байт файла, FTP закрывает соединение данных. Приемник понимает, что это означает, что последние принятые байты являются последними байтами переданного файла.

Управление соединением

Программы-клиенты FTP используют управляющее соединение, чтобы посылать команды и получать ответы от сервера. Обычно команды передаются через управляющее соединение, запрашивая сервер исполнить некоторые связанные с файлами действия на сервере или передать информацию через соединение данных. Клиент FTP создает управляющее соединение тем же образом, что и другие клиенты, рассмотренные в этой книге. Другими словами, клиент создает сокет и соединяет его с официальным портом сервера. Клиент посылает команды серверу через управляющее соединение подобно клиентам SMTP и POP3, передающим команды своим серверам.

Установление соединения данных

Основанные на протоколе FTP программы используют соединение данных для трех основных целей:

• Чтобы послать список файлов или каталогов от сервера клиенту.

• Чтобы послать файл от клиента серверу.

• Чтобы послать файл от сервера клиенту.

Серверы FTP используют соединение данных, чтобы послать списки файлов клиенту FTP. Хотя сервер мог бы использовать многострочный ответ на запрос выдачи списка файлов, соединение данных обладает парой преимуществ. Во-первых, реализация FTP может ограничивать число строк, которые могут включать многострочный ответ. Во-вторых, посылка списка файлов через соединение данных делает более легким для работающего за терминалом пользователя FTP захват и сохранение принятого списка в файл. Когда клиент или сервер использует соединение данных для передачи файлов (или другой информации, типа списка файлов), они обычно следуют процедуре, описанной ниже. Сначала клиент создает соединение данных. Так как клиент инициирует все команды, которые требуют использования соединения данных, он должен также создать само соединение.

В действительности, клиент и сервер обмениваются сообщениями через сокет, соединенный с сервером через официальный порт протокола. Управляющее соединение FTP остается открытым в течение всего сеанса связи клиента с сервером. Однако FTP-клиент создает и поддерживает соединение данных только на протяжении операции передачи данных. После того как операция передачи заканчивается, клиент закрывает соединение данных. Другими словами, FTP-клиент поддерживает соединение данных только на время операции передачи. Каждый раз, когда клиент должен обменяться данными с сервером (через соединение данных), он создает новое соединение. Обратите внимание на то, что передача данных FTP не происходит через официальный порт. Она происходит через порт, который выбирает сетевой компьютер клиента.

Таким образом, FTP-клиент должен выполнить пассивное открытие сокета соединения данных и затем сообщить серверу, какой порт на компьютере клиента он должен использовать, чтобы установить соединение. Иначе говоря, сервер FTP не имеет никаких предположений относительно того, через какой порт посылать клиенту данные, запрошенные по управляющему соединению. После того как клиент сообщает FTP-серверу, какой порт протокола использовать, сервер исполняет активное открытие и использует сокет и порт протокола FTP-клиента, указанный компьютером клиента. Другими словами, для соединения данных FTP-клиент действует подобно серверу. Клиент создает сокет, связывает его с местным адресом, сообщает серверу, какой адрес использовать, чтобы войти в контакт, и ожидает входящее соединение. Однако различие между FTP-клиентом и настоящим сервером в том, что первый принимает соединение только от FTP-сервера на другом конце управляющего соединения. Как вы помните из материала предыдущих глав, сокет сервера хранит маску для адреса удаленного компьютера. Другими словами, сокет сервера обслуживает запросы на установление соединения, пришедшие от любого удаленного компьютера. FTP-клиент хранит адрес FTP-сервера в сокете, созданном для соединения данных. Таким образом, сокет будет принимать запрос на установление соединения только от FTP-сервера. Один и тот же процесс создания соединения данных происходит независимо от того, хочет клиент послать или получить файл. В обоих случаях клиент создает сокет, связывает его с местным адресом, сообщает FTP-серверу, какой порт использовать для контакта, и ждет запроса на установление соединения от FTP-сервера. Другими словами, и в случае посылки, и в случае получения файлов FTP-клиент исполняет пассивное открытие, а FTP-сервер исполняет активное открытие соединения.

Команды FTP

В протоколе передачи файлов определено более тридцати команд, которые программа-клиент может использовать для управления сервером. FTP-команды делятся на три категории: команды контроля доступа, команды передачи параметров и команды обслуживания. Команды контроля доступа передают информацию, идентифицирующую пользователя серверу или сообщает серверу, к каким каталогам программа-клиент желает получить доступ. Команды, передающие параметры, позволяют клиенту определять опции FTP, уже рассмотренные нами: регистрируют тип, формат файла, структуру файла и режим передачи. Команды обслуживания FTP задают выполнение операций по передаче файлов.