МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Белгородский государственный технологический университет
им. В.Г. Шухова

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Утверждено
научно-методическим советом
университета

Сети электронно-вычислительных машин и телекоммуникации

Методические указания к выполнению лабораторных работ
для студентов 4 курса направления бакалавриата
230100.62 – Программная инженерия

Белгород
2014

УДК 004.7 (07)
ББК 32.973.202 я7
С 33

Составители: ст. преп. *Е.А. Федотов*
ст. преп. *А.И. Гарибов*

Рецензент: канд. техн. наук, доцент *В.Г. Синюк*

С 33 Сети электронно-вычислительных машин и телекоммуникации: методические указания к выполнению лабораторных работ / сост. Е. А. Федотов, А.И. Гарибов. – Белгород: Изд-во БГТУ, 2014. – 77 с.

В методические указания включены требования и рекомендации к выполнению лабораторных работ по дисциплине «Сети электронно-вычислительных машин и телекоммуникации», а также задания для выполнения данных работ.

Методические указания предназначены для студентов 4 курса направления бакалавриата 230100.62 – Программная инженерия. Данное издание публикуется в авторской редакции.

УДК 004.7 (07)
ББК 32.973.202 я7

© Белгородский государственный
технологический университет
(БГТУ) им. В.Г. Шухова, 2014

Оглавление

Лабораторная работа № 1

Протокол сетевого уровня IPX

Цель работы: изучить протокол сетевого уровня IPX, основные функции API драйвера IPX и разработать программу для приема/передачи данных.

Краткие теоретические сведения

Протокол IPX – это протокол сетевого уровня модели взаимодействия открытых систем (OSI) реализующий передачу пакетов (сообщений) между станциями сети на уровне датаграмм. Датаграмма – это сообщение, доставка которого получателю не гарантируется. Следовательно, для обеспечения надежной работы нужно предусмотреть схему уведомления других станций о том, что переданные ими пакеты успешно приняты и обработаны. Более того, последовательность отправления пакетов передающим узлом может отличаться от последовательности приема этих пакетов, что также необходимо учитывать [1, 3, 7].

В процессе обмена сообщениями на уровне сеанса связи участвуют только две станции сети. На уровне датаграмм есть возможность посылать сообщение одновременно всем станциям сети.

Система адресов, используемая в протоколе IPX, представлена несколькими компонентами: это номер сети, адрес станции в сети и идентификатор программы на рабочей станции.

Номер сети - это номер сегмента сети, определяемого системным администратором. Если в общей сети есть мосты, каждая отдельная сеть, подключенная через мост, должна иметь свой, уникальный номер сети.

Адрес станции - это число, которое является уникальным для каждой рабочей станции. При использовании адаптеров Ethernet уникальность обеспечивается изготовителем сетевого адаптера. Специальный адрес FFFFFFFFFFFFh используется для рассылки данных всем станциям данной сети одновременно.

Идентификатор программы на рабочей станции (сокет) - число, которое используется для адресации определенной программы,

работающей на компьютере. В среде мультизадачных операционных систем, на каждой рабочей станции в сети одновременно может быть запущено несколько приложений. Для того, чтобы послать данные конкретной программе, используется идентификация программ при помощи сокетов. Каждая программа, желающая принимать или передавать данные по сети, должна получить свой, уникальный для данной рабочей станции, идентификатор - сокет.

Формат передаваемых с использованием протокола IPX по сети пакетов представлен на рис. 1.1.

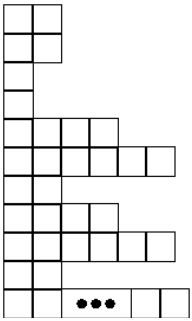

Рис. 1.1. Структура пакета IPX

Все поля, кроме поля Data, представляют собой заголовок пакета. В заголовке располагаются: адрес назначения, обратный адрес и некоторая служебная информация. Все поля заголовка содержат значения в перевернутом формате, т. е. по младшему адресу записывается младший байт данных.

Поле **Checksum** предназначено для хранения контрольной суммы передаваемых пакетов. При формировании пакетов не нужно заботиться о содержимом этого поля, так как проверка данных по контрольной сумме выполняется драйвером сетевого адаптера.

Поле **Length** определяет общий размер пакета. Длина заголовка фиксирована и равна 30 байт. Размер передаваемых в поле **Data** данных может составлять от 0 до 546 байт. При формировании пакетов не нужно проставлять длину пакета в поле **Length**, протокол IPX делает это сам.

Поле **TransportControl** является счетчиком мостов, которые проходит пакет на своем пути от передающей станции к принимающей. Каждый раз, когда пакет проходит через мост, значение этого счетчика увеличивается на единицу. IPX перед передачей пакета сбрасывает содержимое этого поля в нуль. Так как IPX сам следит за содержимым этого поля, при формировании пакетов не нужно изменять или устанавливать его в какое-либо состояние.

Поле **PacketType** определяет тип передаваемого пакета. Для IPX следует установить значение равное 4.

Поле **DestNetwork** определяет номер сети, в которую передается пакет.

Поле **DestNode** определяет адрес рабочей станции, которой предназначен пакет.

Поле **DestSocket** предназначено для адресации программы, запущенной на рабочей станции, которая должна принять пакет.

Поля **SourceNetwork, SourceNode** и **SourceSocket** содержат соответственно номер сети, из которой посылается пакет, адрес передающей станции и сокет программы, передающей пакет.

Поле **Data** содержит передаваемые данные.

Прикладные программы все свои запросы на прием и передачу пакетов направляют драйверу IPX, который, в свою очередь, обращается к драйверу сетевого адаптера. Для приема или передачи пакета прикладная программа должна подготовить пакет данных, сформировав его заголовок, и построить так называемый блок управления событием ECB (Event Control Block). В блоке ECB задается адресная информация для передачи пакета, адрес передаваемого пакета в оперативной памяти и некоторая другая информация. Подготовив блок ECB, прикладная программа передает его адрес соответствующей функции IPX для выполнения операции приема или передачи пакета.

Функции IPX, принимающие или передающие пакет, не выполняют ожидания завершения операции, а сразу возвращают управление вызвавшей их программе. Прием или передача выполняются сетевым адаптером автономно и асинхронно по

отношению к программе, вызвавшей функцию IPX для передачи данных. После того, как операция передачи данных завершилась, в соответствующем поле блока ECB устанавливается признак завершения операции. Программа может периодически проверять ECB для обнаружения этого признака.

Кроме того, в блоке ECB следует указать адрес процедуры, которая будет вызвана при завершении выполнения операции передачи данных. Этот способ предпочтительнее, так как прикладная программа не будет тратить время на периодическую проверку блока ECB.

Формат блока ECB представлен на рис. 1.2.

Блок ECB состоит из фиксированной части размером 36 байт и массива дескрипторов, описывающих отдельные фрагменты передаваемого или принимаемого пакета данных.

Поле **Link** предназначено для организации списков, состоящих из блоков ECB. Драйвер IPX использует это поле для объединения переданных ему блоков ECB в списки, записывая в него полный адрес в формате [сегмент : смещение]. После того, как IPX выполнит данную ему команду и закончит все операции над блоком ECB, программа может распоряжаться полем Link по своему усмотрению. В частности, она может использовать это поле для организации списков или очередей свободных или готовых для чтения блоков ECB.

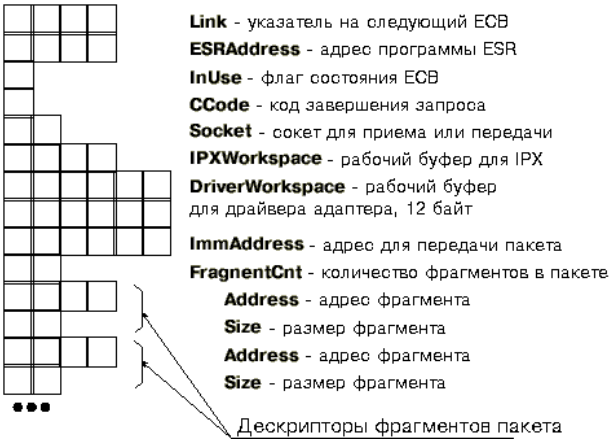

Рис. 1.2. Формат блока ECB

Поле **ESRAddress** содержит полный адрес программного модуля (в формате [сегмент : смещение]), который получает управление при завершении процесса чтения или передачи пакета IPX. Этот модуль называется программой обслуживания события ESR (Event Service Routine). Если программа не использует ESR, она должна записать в поле **ESRAddress** нулевое значение. В этом случае о завершении выполнения операции чтения или передачи можно узнать по изменению содержимого поля **InUse**.

Поле **InUse** может служить индикатором завершения операции приема или передачи пакета. Перед тем как вызвать функцию IPX, программа записывает в поле **InUse** нулевое значение. Пока операция передачи данных, связанная с данным ECB, не завершилась, поле **InUse** содержит следующие значения:

- FFh - ECB используется для передачи пакета данных;
- FEh - ECB используется для приема пакета данных, предназначенного программе с конкретным сокетом;
- FDh - ECB используется функциями асинхронного управления событиями AES (Asynchronous Event Sheduler), ECB находится в состоянии ожидания истечения заданного временного интервала;
- FBh - пакет данных принят или передан, но ECB находится во внутренней очереди IPX в ожидании завершения обработки.

Программа может постоянно опрашивать поле InUse, ожидая завершения процесса передачи или приема данных. Как только в этом поле окажется нулевое значение, программа считает, что запрошенная функция выполнена. Результат выполнения можно получить в поле CCode, где после выполнения функции IPX содержится код результата выполнения.

Если с данным ECB была связана команда приема пакета, в поле **CCode** могут находиться следующие значения:

- 00 - пакет был принят без ошибок;
- FFh - указанный в ECB сокет не был предварительно открыт программой;

- FDh - переполнение пакета: либо поле количества фрагментов в пакете FragmentCnt равно нулю, либо буферы, описанные дескрипторами фрагментов, имеют недостаточный размер для записи принятого пакета;
- FCh - запрос на прием данного пакета был отменен специальной функцией драйвера IPX.

Если ECB использовался для передачи пакета, в поле **CCode** после завершения передачи могут находиться следующие значения:

- 00 - пакет был передан без ошибок (но это не означает, что пакет был доставлен по назначению и успешно принят станцией-адресатом);
- FFh - пакет невозможно передать физически из-за неисправности в сетевом адаптере или в сети;
- FEh - пакет невозможно доставить по назначению, так как станция с указанным адресом не существует или неисправна;
- FDh - пакет сбойный, т.е. либо имеет длину меньше 30 байт, либо первый фрагмент пакета по размеру меньше размера стандартного заголовка пакета IPX, либо поле количества фрагментов в пакете FragmentCnt равно нулю;
- FCh - запрос на передачу данного пакета был отменен специальной функцией драйвера IPX.

Поле **Socket** содержит номер сокета, связанный с данным ECB. Если ECB используется для приема, это поле содержит номер сокета, на котором выполняется прием пакета. Если же ECB используется для передачи, это поле содержит номер сокета передающей программы.

Поле **IPXWorkspace** зарезервировано для использования драйвером IPX. Приложение не должно инициализировать или изменять содержимое этого поля, пока обработка ECB не завершена.

Поле **DriverWorkspace** зарезервировано для использования драйвером сетевого адаптера. Программа не должна инициализировать или изменять содержимое этого поля, пока обработка ECB не завершена.

Поле **ImmAddress** (Immediate Address - непосредственный адрес) содержит адрес узла в сети, в который будет направлен пакет. Если

пакет передается в пределах одной сети, поле **ImmAddress** будет содержать адрес станции-получателя (такой же, как и в заголовке пакета IPX). Если же пакет предназначен для другой сети и будет проходить через мост, поле **ImmAddress** будет содержать адрес этого моста в сети, из которой передается пакет.

Поле **FragmentCnt** содержит количество фрагментов, на которые нужно разбить принятый пакет, или из которых необходимо собрать передаваемый пакет. Механизм фрагментации позволяет избежать пересылок данных или непроизводительных потерь памяти. Можно указать отдельные буферы для приема данных и заголовка пакета. Если принимаемые данные имеют какую-либо структуру, можно рассредоточить отдельные блоки по соответствующим буферам. Значение, записанное в поле FragmentCnt, не должно быть равно нулю. Если в этом поле записано значение 1, весь пакет вместе с заголовком записывается в один общий буфер.

Далее располагаются дескрипторы фрагментов, состоящие из указателя в формате [сегмент : смещение] на фрагмент **Address** и поля размера фрагмента **Size**.

Если программе нужно разбить принятый пакет на несколько частей, то нужно установить в поле **FragmentCnt** значение, равное количеству требуемых фрагментов. Д для каждого фрагмента необходимо создать дескриптор, в котором указать адрес буфера и размер фрагмента. Аналогичные действия выполняются и при сборке пакета перед передачей из нескольких фрагментов [1, 2].

Основные функции API для работы с протоколом IPX

Для того чтобы можно было работать в сети с протоколом IPX, сначала необходимо проверить, установлен ли драйвер этого протокола. Затем необходимо получить адрес вызова этого драйвера - точку входа API. В дальнейшем можно будет вызывать драйвер при помощи команды межсегментного вызова процедуры по адресу точки входа API драйвера IPX.

Для того чтобы проверить, загружен ли драйвер IPX, необходимо загрузить в регистр AX значение 7A00h и вызвать мультиплексное

прерывание INT 2Fh. Если после возврата из прерывания в регистре AL будет значение FFh, то драйвер IPX загружен. Адрес точки входа для вызова API драйвера при этом будет находиться в регистровой паре ES:DI. Если же содержимое регистра AL после возврата из прерывания INT 2Fh будет не равно FFh, то драйвер IPX/SPX не загружен. Это означает, что на данной рабочей станции не загружены резидентные программы ipx.exe или ipxodi.exe, обеспечивающие API для работы с протоколами IPX и SPX. Для вызова API в регистр BX необходимо загрузить код выполняемой функции. Значения, загружаемые в другие регистры, зависят от выполняемой функции.

Для работы с IPX используются следующие основные функции.

Функция **IPXOpenSocket** предназначена для получения сокетов.

На входе BX = 00h, AL = Тип сокета: 00h - короткоживущий; FFh -долгоживущий, DX = Запрашиваемый номер сокета или 0000h, если требуется получить динамический номер сокета. Байты номера сокета находятся в перевернутом виде.

На выходе AL = Код завершения (00h - сокет открыт; FFh - этот сокет уже был открыт раньше; FEh - переполнилась таблица сокетов), DX = Присвоенный номер сокета.

Функция **IPXCloseSocket** предназначена для освобождения сокетов, т.к. сокеты являются ограниченным ресурсом. На входе BX = 0lh, DX = Номер закрываемого сокета. На выходе регистры не используются.

Функция **IPXGetLocalTaget** применяется для вычисления значения непосредственного адреса, помещаемого в поле ImmAddress блока ECB перед передачей пакета.

На входе BX = 09h, ES:SI = Указатель на буфер длиной 10 байт, в который будет записан адрес станции, на которой работает данная программа. Адрес состоит из номера сети и адреса станции в сети. На выходе регистры не используются.

Станция-получатель может находиться в другой сети, поэтому прежде чем достигнуть цели, пакет может пройти один или несколько мостов. Поле непосредственного адреса ImmAddress блока ECB должно содержать либо адрес станции назначения (если передача

происходит в пределах одной сети), либо адрес моста (если пакет предназначен для рабочей станции, расположенной в другой сети). Используя, указанный в буфере размером 12 байт, полный сетевой адрес, состоящий из номера сети, адреса станции в сети и сокета приложения, функция IPXGetLocalTaget вычисляет непосредственный адрес, т.е. адрес той станции в данной сети, которая получит передаваемый пакет.

Формат полного адреса представлен на рис. 1.3.

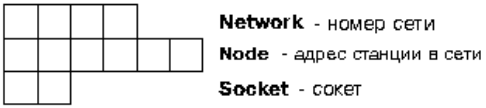

Рис. 1.3. Формат полного адреса

В поле Network указывается номер сети, в которой расположена станция, принимающая пакет.

Поле Node должно содержать адрес станции в сети с номером, заданным содержимым поля Network. Если пакет должны принять все станции, находящиеся в сети Network, в поле Node необходимо записать адрес FFFFFFFFFFFFh.

Поле Socket адресует конкретную программу, работающую на станции с заданным адресом.

Если программа-сервер принимает пакеты от клиентов и возвращает клиентам свои пакеты, нет необходимости пользоваться функцией IPXGetLocalTaget для заполнения поля ImmAddress блока ECB перед отправкой ответа станции-клиенту. Когда от клиента приходит пакет, в поле ImmAddress блока ECB автоматически записывается непосредственный адрес станции (или моста), из которой пришел пакет. Поэтому для отправки ответного пакета можно воспользоваться тем же самым ECB с уже проставленным значением в поле ImmAddress.

Функция **IPXListenForPacket** предназначена для начала процесса приема пакетов данных из сети. На входе BX = 04h, ES:SI = Указатель

на заполненный блок ECB. На выходе регистры не используются. Сразу после вызова функции IPXListenForPackets в поле InUse блока ECB устанавливается значение FEh, которое означает, что для данного блока ECB ожидается прием пакета.

После прихода пакета в поле InUse блока ECB устанавливается значение 0h и вызывается программа ESR. Если ее адрес был задан перед вызовом функции **IPXListenForPacket**, в поле CCode блока ECB драйвер IPX записывает код результата приема пакета, а в поле ImmAddress - непосредственный адрес станции, из которой пришел пакет. Если пакет пришел из другой сети, в этом поле будет стоять адрес моста.

Функция **IPXSendPacket** подготавливает блок ECB и связанный с ним заголовок пакета для передачи пакета по сети. На входе BX = 03h, ES:SI = Указатель на заполненный блок ECB. На выходе регистры не используются.

Сразу после вызова функции **IPXSendPacket**, IPX-служба ставит блок ECB в очередь на передачу и возвращает управление вызвавшей ее программе, не дожидаясь прихода пакета, в поле InUse блока ECB устанавливается значение FFh. Сама передача пакета происходит асинхронно по отношению к вызывавшей ее программе.

После завершения процесса передачи пакета в поле InUse записывается значение 0h, в поле CCode блока ECB – код результата приема пакета. Результат выполнения передачи пакета можно узнать, проанализировав поле CCode блока ECB.

Функция **IPXRelinquishControl** выделяет драйверу IPX процессорное время, необходимое для его правильной работы. Если программа не использует ESR, она, должна в цикле опрашивать поле InUse блока ECB, для которого выполняется ожидание завершения процесса приема или передачи пакета. Однако для правильной работы драйвера IPX в цикл ожидания необходимо вызовать эту функцию.

Функция **IPXCancelEvent** отменяет ожидание приема или отправки пакета, связанного с блоком ECB. Ее рекомендуется выполнять при аварийном выходе из программы для каждого подготовленного, но ещё неиспользованного блока ECB.

В структуре ECB необходимо заполнить следующие поля:

- **ESRAddress** – адрес процедуры ESR, либо нулевое значение, если процедура ESR не должна вызываться после передачи пакета;
- **Socket** – номер сокета;
- **ImmAddress** – адрес узла в сети, которому будет отправлен пакет;
- **FragmentCnt** – число фрагментов, на которые нужно разбить отправляемый пакет;
- дескрипторы фрагментов – указатели на буферы фрагментов **Address** и размеры фрагментов **Size**.

Также необходимо подготовить заголовок передаваемого пакета и заполнить следующие поля:

- **PacketType –** определяет тип передаваемого пакета; для передачи пакетов средствами IPX необходимо установить в поле PacketType значение 4;
- **DestNetwork –** определяет номер сети, в которую передается пакет;
- **DestNode –** определяет адрес рабочей станции, которой предназначен пакет;
- **DestSocket –** определяет номер сокета принимающей программы на станции получателя;
- **SourceNetwork, SourceNode, SourceSocket –** соответственно номер сети, из которой посылается пакет, адрес передающей станции и сокет программы, передающей пакет.

Следует отметить, что передавать пакеты можно "самому себе", т.е. передающая и принимающая программы могут работать на одной и той же станции и использовать один и тот же сокет. Если при отправке или приеме пакетов не используется процедура ESR, а контроль за выполнением IPX-службы выполняется с помощью анализа значения поля InUse в ECB, то в цикле просмотра необходимо выделять службам IPX процессорное время, необходимое для работы. Это делается с помощью процедуры IPXRelinquishControl. Если программа подготовила для приема пакетов несколько блоков ECB, то

для приема пришедшего пакета будет использован один из подготовленных ECB. Однако не гарантируется, что блоки ECB будут использоваться в том порядке, в котором они ставятся на ожидание приема функцией IPXListenForPacket.

Кроме того, если программа подготовила для передачи пакетов несколько блоков ECB в некотором порядке, то это не означает, что они будут приняты станцией получателем в том же самом порядке. Если свободных блоков ECB, ожидающих прием пакета, нет, то приходящий пакет будет проигнорирован. Если ожидается/отправляется пакет по данному сокету, а сокет не открыт, пришедший/отправленный пакет также будет проигнорирован.

Задание к работе

1. Разработать программу “Сервер”, которая посылает клиентам сети файл с использованием протокола IPX в среде DOS на языке программирования Pascal или C.

2. Разработать программу “Клиент”, которая принимает от сервера файл на языке программирования Pascal или C.

3. Провести анализ функционирования разработанных программ при передаче файла в формате *.jpg размером не менее 1 Мб (одновременная работа 2-х, 3-х и т.д. приложений на 2-х, 3-х и т.д. компьютерах ЛВС), сделать выводы.

Содержание отчета

1. Краткие теоретические сведения.
2. Основные функции API, использованные в данной работе.
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Контрольные вопросы

1. Система адресации в протоколе IPX.
2. Формат заголовка пакета, передаваемого через протокол IPX.
3. Что представляет собой блок управления событием ECB? С какой целью он используется?
4. Опишите 2 способа, с помощью которых в программе можно узнать о завершении операций приёма или передачи.
5. Какие функции API и в каком порядке нужно вызвать для начала операции отправки пакетов?
6. Недостатки протокола IPX.
7. Сколько байт данных максимально можно передать в одном пакете?
8. Каким образом можно выполнить рассылку пакета всем рабочим станциям сети?

Лабораторная работа № 2

Программирование протоколов IPX/SPX с использованием библиотеки Winsock

Цель работы: изучить протоколы IPX/SPX, основные функции библиотеки Winsock и разработать программу для приема/передачи пакетов.

Краткие теоретические сведения

Протокол IPX (Internetwork Packet Exchange) является оригинальным протоколом сетевого уровня стека Novell, разработанным в начале 80-х годов на основе протокола Internetwork Datagram Protocol (IDM) компании Xerox [1, 3, 7].

Протокол IPX соответствует сетевому уровню модели OSI и поддерживает только дейтаграммный (без установления соединений)

способ обмена сообщениями. В сети NetWare самая быстрая передача данных при наиболее экономном расходовании памяти реализуется именно протоколом IPX.

Для надежной передачи пакетов используется протокол транспортного уровня SPX (Sequenced Packet Exchange), который работает с установлением соединения и восстанавливает пакеты при их потере или повреждении. Если по каким-то причинам пакет не дошел до получателя, выполняется его повторная передача. Следовательно, последовательность отправления совпадает с последовательностью получения пакетов. Обмен пакетами на уровне сеанса связи реализован с помощью протокола SPX, который построен на базе IPX.

Фирма Novell в сетевой операционной системе NetWare применяла протокол IPX для обмена датаграммами и протокол SPX для обмена в сеансах.

Для некоторых приложений (например, для программ, передающих файлы между рабочими станциями) удобнее использовать сетевой протокол более высокого уровня, обеспечивающий гарантированную доставку пакетов в правильной последовательности. Разумеется, программа может сама следить за тем, чтобы все переданные пакеты были приняты. Однако в этом случае придется делать собственную надстройку над протоколом IPX - собственный протокол передачи данных.

SPX – протокол последовательного обмена пакетами (Sequenced Packet Exchange Protocol), разработанный Novell [6]. Система адресов протокола SPX аналогична системе адресов протокола IPX и также состоит из 3 частей: номера сети, адреса станции и сокета.

Протокол SPX использует такой же блок ECB для передачи и приёма пакетов, что и протокол IPX. Однако, пакет, передаваемый при помощи протокола SPX, имеет более длинный заголовок. Дополнительно к 30 байтам стандартного заголовка пакета IPX добавляется еще 12 байт (рис. 2.1).

Поле **ConnControl** представляет собой как набор битовых флагов, управляющих передачей данных по каналу SPX.

Поле **DataStreamType** состоит из однобитовых флагов, которые используются для классификации данных, передаваемых или принимаемых при помощи протокола SPX.

Поле **SourceConnID** содержит номер канала связи передающей программы, присвоенный драйвером SPX при создании канала связи. Этот номер должен указываться функции передачи пакета средствами SPX.

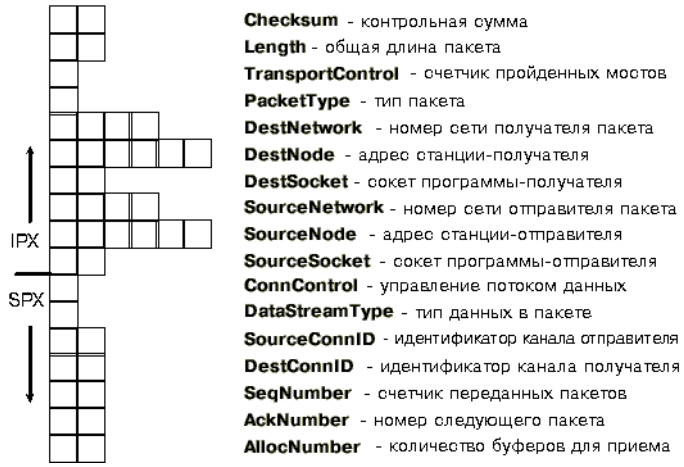

Рис. 2.1. Формат заголовка пакета SPX

Поле **DestConnID** содержит номер канала связи принимающей стороны. Так как все пакеты приходят на один номер сокета и могут принадлежать разным каналам связи (на одном сокете можно открыть несколько каналов связи), необходимо классифицировать приходящие пакеты по номеру канала связи.

Поле **SeqNumber** содержит счетчик пакетов, переданных по каналу в одном направлении. На каждой стороне канала используется свой счетчик. После достижения значения FFFFh счетчик сбрасывается в нуль, после чего процесс счета продолжается. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

Поле **AckNumber** содержит номер следующего пакета, который должен быть принят драйвером SPX. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

Поле **AllocNumber** содержит количество буферов, распределенных программой для приема пакетов. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение [4].

Библиотека Winsock

Windows Sockets API (WSA) (сокр. **Winsock)** – техническая спецификация, которая определяет, как сетевое программное обеспечение Windows будет получать доступ к сетевым сервисам [3].

Winsock – это интерфейс сетевого программирования для Microsoft Windows. Winsock основывается на сокетной парадигме, изложенной в документах под названием Berkley System Distribution от University of California в Berkley.

Основные операционные среды (Unix подобные системы, Windows) базируются в настоящее время на идеологии соединителей (socket). Эта технология была разработана в университете г. Беркли (США) для системы Unix, поэтому соединители иногда называют соединителями Беркли. Соединители реализуют механизм взаимодействия не только партнеров по телекоммуникациям, но и процессов в ЭВМ вообще.

Winsock включает в себя несколько стилей программирования. Первый – это стандартный однопотоковый стиль с блокированием потока определенными командами, второй – с использованием оконных процедур и третий – с использованием асинхронных процедур. Стандартная модель программирования от Berkley является de facto для сетей TCP/IP, но под Windows можно использовать эту библиотеку для программирования протоколов IPX/SPX.

Winsock предназначен для использования во всех версиях MS Windows, начиная с 3.0. Для того чтобы программа могла корректно работать с библиотекой Winsock необходимо проверить версию

библиотеки Winsock, а так же вообще наличие этой библиотеки в системе. Библиотека функции Winsock расположена в файле wsock32.dll (ws2_32.dll для версии 2.0 этой библиотеки) или winsock.dll для 32-бит и 16-бит приложений соответственно. Также, необходимо подключить заголовочные файлы winsock.h (winsock2.h), а для работы с протоколами IPX и SPX еще и заголовочный файл wsipx.h.

Описание функций

Функция **WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData)** инициализирует библиотеку Winsock. В случае успеха возвращает 0. Дальше можно использовать любые остальные функции этой библиотеки, иначе возвращает код возникшей ошибки. WwVersionRequested – это необходимая минимальная версия библиотеки, при присутствии которой приложение будет корректно работать. Младший байт содержит номер версии, а старший – номер ревизии. LpWSAData – структура, в которую возвращается информация по инициализированной библиотеке (статус, версия и т.д.).

Функция **WSAGetLastError (void)** возвращает код ошибки возникшей при выполнении последней операции.

После работы с библиотекой, её необходимо выгрузить из памяти. Функция **WSACleanup (void)** осуществляет очистку памяти, занимаемой библиотекой Winsock. Функция деинициализирует библиотеку Winsock и возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET_ERROR. Расширенный код ошибки можно получить при помощи функции **WSAGetLastError.**

Порядок байт на машинах PC отличается от порядка используемого в сетях, поэтому необходимы некоторые преобразования определенных данных, например номера порта, чтобы он был правильным при использовании функций библиотеки Winsock. Ниже приведены функции преобразования порядка байт:

u_short htons(u_short hostshort);

u_long htonl(u_long hostlong);

u_long ntohl(u_long netlong);

u_short ntohs(u_short netshort);

В качестве параметра передаётся число, которое необходимо преобразовать. Функция возвращает преобразованное число.

Функция **SOCKET (int af, int type, int protocol)** возвращает либо дескриптор созданного сокета, либо ошибку INVALID_SOCKET. Расширенный код ошибки можно получить при помощи функции WSAGetLastError.

Параметр af содержит сведения о семействе протоколов (AF_INET, AF_IPX). В данной лабораторной работе необходимо использовать константу AF_IPX. Параметр type – тип передаваемых данных (поток или дейтаграммы). В данной лабораторной работе для IPX необходимо использовать константу SOCK_DGRAM, а для SPX – константу SOCK_SEQPACKET, которая означает, что пакеты будут отсылаться последовательно и в порядке очереди. Параметр protocol – протокол передачи данных. Для протокола IPX используется константа NSPROTO_IPX, для SPX – NSPROTO_SPX.

Чтобы работать дальше с созданным сокетом его нужно привязать к какому-нибудь локальному адресу и порту. Этим занимается функция **bind (SOCKET s, const struct sockaddr FAR* name, int namelen**). Здесь s – дескриптор сокета, который данная функция именует; name – указатель на структуру имени сокета; namelen – размер, в байтах, структуры name.

В сетях IPX/SPX структура sockaddr имеет вид:

```
typedef struct sockaddr_ipx {
short sa_family;
char  sa_netnum[4];
char  sa_nodenum[6];
unsigned short sa_socket;
} SOCKADDR_IPX, *PSOCKADDR_IPX,FAR *LPSOCKADDR_IPX;
```

Параметры структуры:

- sin_family — семейство протоколов.
- sin_netnum — номер сети.

- sin_nodenum — номер узла
- sa_socket – номер сокета

Если порт установить в 0, то система сама пытается подыскать свободный порт в интервале от 1024 до 5000.

Если в качестве адреса указать константу INADDR_ANY (0) для сетей TCP/IP или 0 в сетях IPX/SPX, то система попытается использовать все доступные адреса для сокета.

Функция **listen (SOCKET s, int backlog)** переводит сокет в состояние “прослушивания” (для протокола SPX). Здесь s – дескриптор сокета; backlog – это максимальный размер очереди входящих сообщений на соединение.

Эта функция используется сервером, чтобы информировать ОС, что он ожидает запросы связи на данном сокете. Без такой функции всякое требование связи с этим сокетом будет отвергнуто [6].

Функция **connect (SOCKET s, const struct sockaddr FAR* name, int namelen)** используется процессом-клиентом для установления связи с сервером по протоколу SPX. В случае успешного установления соединения connect возвращает 0, иначе SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **accept (SOCKET s, struct sockaddr FAR* addr, int FAR* addrlen)** используется для принятия связи на сокет. Здесь s – дескриптор сокета; addr – указатель на структуру sockaddr; addrlen – размер структуры addr. Сокет должен быть уже слушающим в момент вызова функции. Если сервер устанавливает связь с клиентом, то данная функция возвращает новый сокет-дескриптор, через который и производит общение клиента с сервером. Пока устанавливается связь клиента с сервером, функция блокирует другие запросы связи с данным сервером, а после установления связи “прослушивание” запросов возобновляется [8].

В случае автоматического распределения адресов и портов узнать какой адрес и порт присвоен сокету можно при помощи функции **getsockname (SOCKET s, struct sockaddr FAR* name, int FAR* namelen).** Здесь s — дескриптор сокета; name — структура sockaddr, в которую система поместит данные; namelen — размер, в байтах,

структуры name. Если операция выполнена успешно, возвращает 0, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Передача данных по протоколу IPX осуществляется с помощью функции **sendto (SOCKET s, const char FAR * buf, int len, int flags, const struct sockaddr FAR * to, int tolen).** Здесь s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных; to - указатель на структуру sockaddr, которая содержит адрес сокета-приёмника; tolen - размер структуры to. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Передача данных по протоколу SPX осуществляется с помощью функции **send (SOCKET s, const char FAR * buf, int len, int flags).** Здесь s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Прием данных по протоколу IPX осуществляется с помощью функции **recvfrom (SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen)**. Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Прием данных по протоколу SPX осуществляется с помощью функции **recv (SOCKET s, char FAR* buf, int len, int flags).** Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **closesocket(SOCKET s)** служит для закрытия сокета. Возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Рекомендации к разработке программы

Обычно в сети одна из рабочих станций принимает запросы на выполнение каких-либо действий от других рабочих станций. Так как станция обслуживает запросы, она называется сервером (serve - обслуживать, server - обслуживающее устройство). Выполнив запрос, сервер посылает ответ в запросившую станцию, которая называется клиентом. В сети может быть много серверов и много клиентов. Одни и те же клиенты могут посылать запросы разным серверам [5].

Говоря более строго, сервером или клиентом является не рабочая станция, а запущенная на ней программа. В мультизадачной среде разные программы, запущенные одновременно на одной и той же рабочей станции могут являться и клиентами, и серверами.

Программа-сервер, выполнив очередной запрос, переходит в состояние ожидания. Она ждет прихода пакета данных от программы-клиента. В зависимости от содержимого этого пакета программа-сервер может выполнять различные действия, в соответствии с логикой работы программы. Например, она может принять от программы-клиента дополнительные пакеты данных или передать свои пакеты.

Сервер и клиент при необходимости на какое-то время или навсегда могут поменяться местами, изменив свое назначение на противоположное.

Для того, чтобы создавать программы-серверы и программы-клиенты, необходимо научиться выполнять две задачи:

1. инициализацию сервера и клиента;
2. прием и передачу пакетов данных.

Система «клиент – сервер» с использованием протокола IPX должна быть реализована по тому же принципу, что и в лабораторной работе 1.

Система «клиент – сервер» с использованием протокола SPX должна работать по следующему принципу: сервер при запуске демонстрирует пользователю полный сетевой адрес и создаёт поток, вызывающий функцию ожидания подключений. В свою очередь, данный поток для каждого подключившегося клиента создаёт отдельный поток, работающий с ним. Таким образом, сервер способен обслуживать несколько клиентов в параллельном режиме.

Процесс-клиент должен требовать ввода сетевого адреса сервера и имени требуемого файла. Процесс-клиент отправляет имя файла серверу и ждёт ответа. Сервер передаёт файл клиенту, распределяя его по пакетам.

Следует обратить внимание на правильность указания адреса сервера в функции connect при использовании протокола SPX. Тот адрес, который возвращает функция getsockname подходит только для работы приложений на одном компьютере. Для того чтобы сервер и клиент могли обмениваться сообщениями непосредственно по сети с использованием протокола SPX, необходим другой адрес. Чтобы получить этот адрес можно использовать протокол IPX: клиент отправляет широковещательное сообщение; сервер посылает ответ, из которого на стороне клиента можно извлечь адрес, пригодный для последующего сетевого обмена сообщениями по протоколу SPX.

Задание к работе

1. Разработать программу “Сервер” (на языке программирования Pascal или C), которая принимает запросы от клиентов и посылает им в качестве ответа некоторое сообщение.

2. Разработать программу “Клиент” на языке программирования Pascal или C), которая посылает запрос серверу и “ждет” от него ответного сообщения.

3. Провести анализ функционирования разработанных программ при передаче файла в формате *.jpg размером не менее 20 Мб (одновременная работа 2-х, 3-х и т.д. приложений на 2-х, 3-х и т.д. компьютерах ЛВС), сделать выводы.

4. Провести сравнительный анализ протоколов IPX и SPX. Сделать выводы.

Содержание отчета

1. Краткие теоретические сведения.
2. Используемые функции.
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Контрольные вопросы

1. Назовите отличия протокола SPX от IPX.
2. Что представляет собой библиотека Winsock?
3. Какие действия необходимо выполнить для корректного создания сокета, настроенного на приём сообщений?
4. Назовите функции библиотеки WInsocket, используемые для отправки и приёма сообщений через протокол IPX.
5. Что представляет собой структура sockaddr?
6. Принцип построения программы «клиент-сервер» с использованием протокола SPX.

Лабораторная работа № 3

Программирование протокола IP с использованием библиотеки Winsock

Цель работы: изучить принципы и характеристику протокола IP и разработать программу для приема/передачи пакетов с использованием библиотеки Winsock.

Краткие теоретические сведения

Internet Protocol или IP (англ. internet protocol - межсетевой протокол) - маршрутизируемый сетевой протокол сетевого уровня семейства TCP/IP.

Протокол IP используется для негарантированной доставки данных, разделяемых на так называемые пакеты от одного узла сети к другому. Это означает, что на уровне этого протокола (третий уровень сетевой модели OSI) не даётся гарантий надёжной доставки пакета до адресата. В частности, пакеты могут прийти не в том порядке, в котором были отправлены, продублироваться (когда приходят две копии одного пакета - в реальности это бывает крайне редко), оказаться повреждёнными (обычно повреждённые пакеты уничтожаются) или не прибыть вовсе. Гарантии безошибочной доставки пакетов дают протоколы более высокого (транспортного) уровня сетевой модели OSI - например, TCP - который использует IP в качестве транспорта [9].

Обычно в сетях используется IP четвёртой версии, также известный как IPv4. В протоколе IP этой версии каждому узлу сети ставится в соответствие IP-адрес длиной 4 октета (1 октет состоит из 8 бит). При этом компьютеры в подсетях объединяются общими начальными битами адреса. Количество этих бит, общее для данной подсети, называется маской подсети (ранее использовалось деление пространства адресов по классам — A, B, C; класс сети определяется диапазоном значений старшего октета и определяет число адресуемых узлов в данной сети).

IP-пакет представляет собой форматированный блок информации, передаваемый по вычислительной сети. Соединения вычислительных сетей, которые не поддерживают пакеты, такие как традиционные соединения типа «точка-точка» в телекоммуникациях, просто передают данные в виде последовательности байтов, символов или битов. При использовании пакетного форматирования сеть может передавать длинные сообщения более надежно и эффективно.

Структура IP адреса

IP-адрес имеет длину 4 байта и обычно записывается в виде четырех чисел, представляющих значения каждого байта в десятичной форме, и разделенных точками, например 128.10.2.30 – традиционная десятично-точечная форма представления адреса, 10000000 00001010 00000010 00011110 - двоичная форма представления этого же адреса.

Классы сетей IP

IP-адреса разделяются на 5 классов: A, B, C, D, E. Адреса классов A, B и C делятся на две логические части: номер сети и номер узла. На рис. 3.1 показана структура IP-адреса разных классов.

Рис. 3.1. Структура IP-адреса разных классов

Идентификатор сети, также называемый адресом сети, обозначает один сетевой сегмент в более крупной объединенной сети, использующей протокол TCP/IP. IP-адреса всех систем, подключенных к одной сети, имеют один и тот же идентификатор сети. Этот идентификатор также используется для уникального обозначения каждой сети в более крупной объединенной сети.

Идентификатор узла, также называемый адресом узла, определяет узел TCP/IP (рабочую станцию, сервер, маршрутизатор или другое устройство) в пределах каждой сети. Идентификатор узла уникальным образом обозначает систему в том сегменте сети, к которой она подключена.

У адресов класса A старший бит установлен в 0. Длина сетевого префикса 8 бит. Для номера узла выделяется 3 байта (24 бита). Таким

образом, в классе А может быть 126 сетей (2^7 - 2, два номера сети имеют специальное значение). Каждая сеть этого класса может поддерживать максимум 16777214 узлов (2^{24} - 2). Адресный блок класса А может содержать максимум 231 уникальных адресов, в то время как в протоколе IP версии 4 возможно существование 232 адресов. Таким образом, адресное пространство класса А занимает 50% всего адресного пространства протокола IP версии 4. Адреса класса А предназначены для использования в больших сетях, с большим количеством узлов. На данный момент все адреса класса А распределены.

У адресов класса В два старших бита установлены в 1 и 0 соответственно. Длина сетевого префикса – 16 бит. Поле номера узла тоже имеет длину 16 бит. Таким образом, число сетей класса В равно 16384 (2^{14}); каждая сеть класса В может поддерживать до 65534 узлов (2^{16} - 2). Адресный блок сетей класса Класс В предназначен для применения в сетях среднего размера.

У адресов класса С три старших бита установлены в 1, 1 и 0 соответственно. Префикс сети имеет длину 24 бита, номер узла - 8 бит. Максимально возможное количество сетей класса С составляет 2097152 (2^{21}). Каждая сеть может поддерживать максимум 254 узла (2^8 - 2). Класс С предназначен для сетей с небольшим количеством узлов.

Адреса класса D представляют собой специальные адреса, не относящиеся к отдельным сетям. Первые 4 бита этих адресов равны 1110. Таким образом, значение первого октета этого диапазона адресов находится в пределах от 224 до 239. Адреса класса D используются для многоадресных пакетов, с помощью которых во многих разных протоколах данные передаются многочисленным группам узлов. Эти адреса можно рассматривать как заранее запрограммированные в логической структуре большинства сетевых устройств. Это означает, что при обнаружении в пакете адреса получателя такого типа устройство на него обязательно отвечает. Например, если один из хостов передает пакет с IP-адресом получателя 224.0.0.5, на него отвечают все маршрутизаторы (использующие протокол OSPF), которые находятся в сегменте сети с этим адресом Ethernet.

Адреса в диапазоне 240.0.0.0 - 255.255.255.255 называются адресами класса E. Первый октет этих адресов начинается с битов 1111. Эти адреса зарезервированы для будущих дополнений в схеме адресации IP. Но возможность того, что эти дополнения когда-либо будут приняты, находится под вопросом, поскольку уже появилась версия 6 протокола IP (IPv6).

Служебные IP-адреса

Некоторые IP-адреса являются зарезервированными. Для таких адресов существуют следующие соглашения об их особой интерпретации:

1. Если все биты IP-адреса установлены в нуль, то он обозначает адрес данного устройства.
2. Если в поле номера сети стоят нули, то считается, что получатель принадлежит той же самой сети, что и отправитель.
3. Если все биты IP-адреса установлены в единицу, то пакет с таким адресом должен рассылаться всем узлам, находящимся в той же сети, что и отправитель. Такая рассылка называется ограниченным широковещательным сообщением.
4. Если все биты номера узла установлены в нуль, то пакет предназначен для данной сети.
5. Если все биты в поле номера узла установлены в единицу, то пакет рассылается всем узлам сети с данным номером сети. Такая рассылка называется широковещательным сообщением.
6. Если первый октет адреса равен 127, то адрес обозначает тот же самый узел. Такой адрес используется для взаимодействия процессов на одной и той же машине (например, для целей тестирования). Этот адрес имеет название возвратного.

Маска подсети

Поля номеров сети и подсети образуют расширенный сетевой префикс. Для выделения расширенного сетевого префикса используется маска подсети (subnet mask).

Маска подсети – это 32-разрядное двоичное число, в разрядах расширенного префикса содержащая единицу; в остальных разрядах находится ноль. Расширенный сетевой префикс получается побитным сложением по модулю два (операция XOR) IP-адреса и маски подсети.

При таком построении очевидно, что число подсетей представляет собой степень двойки - 2^n, где n - длина поля номера подсети. Таким образом, характеристики IP-адреса полностью задаются собственно IP-адресом и маской подсети.

Стандартные маски подсетей для классов A, B, C приведены в табл. 3.1.

Таблица 3.1.

Стандартные маски подсетей для классов A, B, C

Класс адреса	Биты маски подсети	Маска подсети
Класс A	11111111 00000000 00000000 00000000	255.0.0.0
Класс B	11111111 11111111 00000000 00000000	255.255.0.0
Класс C	11111111 11111111 11111111 00000000	255.255.255.0

Для упрощения записи применяют следующую нотацию (так называемая CIDR-нотация): IP-адрес/длина расширенного сетевого префикса. Например, адрес 192.168.0.1 с маской 255.255.255.0 будет в данной нотации выглядеть как 192.168.0.1/24 (24 – это число единиц, содержащихся в маске подсети).

Функции библиотеки Winsock для работы с протоколом IP

Функция **WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData)** необходима для инициализации библиотеки Winsock. Здесь wVersionRequested – запрашиваемая версия winsock; lpWSAData – структура, в которую возвращается информация

по инициализированной библиотеке (статус, версия и пр.). В случае успеха возвращает 0, иначе возвращает код возникшей ошибки.

Функция **WSAGetLastError (void)** возвращает код ошибки возникшей при выполнении последней операции.

Функция **WSACleanup (void)** очищает память, занимаемую библиотекой Winsock. Возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **u_short htons (u_short hostshort)** осуществляет перевод целого короткого числа из порядка байт, принятого на компьютере, в сетевой порядок байт. hostshort – число, которое необходимо преобразовать. Возвращает преобразованное число [11].

Функция **socket (int af, int type, int protocol)** нужна для создания и инициализации сокета. Здесь af – сведения о семействе адресов. Для интернет-протоколов указывается константа AF_INET; type – тип передаваемых данных (поток или дейтаграммы). В данной лабораторной работе используется константа SOCK_DGRAM; protocol – протокол передачи данных. Для протокола IP используется константа IPPROTO_IP. Функция возвращает дескриптор созданного сокета.

Функция **bind (SOCKET s, const struct sockaddr FAR* name, int namelen)** привязывает адрес и порт к ранее созданному сокету Здесь s – дескриптор сокета; name – структура, содержащая нужный адрес и порт; namelen – размер, в байтах, структуры name.

Функция **unsigned long inet_addr (const char FAR *cp)** конвертирует строку в значение, которое можно использовать в структуре sockaddr_in. Здесь cp – строка, которая содержит IP адрес в десятично-точечном формате (например, 123.23.45.89). Возвращает IP адрес в виде целого числа, либо если произошла ошибка возвращает константу INADDR_NONE.

Для конвертации адреса в стандартный формат используется функция **char (FAR * inet_ntoa(struct in_addr in)**; in – IP-адрес, заданный в сетевом порядке расположения байт. Она возвращает

строку, содержащую IP-адрес в стандартном строчном виде, с числами и точками.

Для определения IP адреса по имени используется функция **struct hostent FAR * gethostbyname (const char FAR * name)**. В качестве результата, функция возвращает структуру hostent.

В случае автоматического распределения адресов и портов узнать какой адрес и порт присвоен сокету можно при помощи функции **getsockname (SOCKET s, struct sockaddr FAR* name, int FAR* namelen).** Если операция выполнена успешно, возвращает 0, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **sendto (SOCKET s, const char FAR * buf, int len, int flags, const struct sockaddr FAR * to, int tolen)** служит для передачи данных. Здесь s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных; to - указатель на структуру sockaddr, которая содержит адрес сокета-приёмника; tolen - размер структуры to. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **recvfrom (SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen)** служит для приема данных. Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **closesocket (SOCKET s)** нужна для закрытия сокета. Она возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Рекомендации к разработке программы

Для начала работы следует подключить необходимые библиотеки и заголовки: #include "winsock.h" либо, #include "winsock2.h" – в зависимости от того, какая версия Winsock будет использоваться. Так же в проект должны быть включены все соответствующие lib-файлы (Ws2_32.lib или Wsock32.lib).

Далее для инициализации Winsock вызываем функцию WSAStartup. При ошибке функция возвращает SOCKET_ERROR. В таком случае можно получить расширенную информацию об ошибке используя вызов WSAGetLastError(). Данная функция возвращает код ошибки (тип int).

Далее необходимо создать сокет. С точки зрения WinsockAPI, сокет — это дескриптор, который может получать или отправлять данные. На практике всё выглядит так: создаём сокет с определёнными свойствами и используем его для подключения, приёма/передачи данных и т.п. Создавая сокет нужно указать его параметры: сокет использует TCP/IP протокол или IPX (если TCP/IP, то какой тип и т.д.). Так как лабораторная работа ориентирована на IP протокол, то остановимся на особенностях сокетов использующих этот протокол.

Мы можем создать два основных типа сокетов работающих по TCP/IP протоколу - SOCK_STREAM и SOCK_DGRAM. Разница в том, что для первого типа сокетов (их еще называют TCP или connection-based socket), для отправки данных сокет должен постоянно поддерживать соединение с адресатом, при этом доставка пакета адресату гарантирована. Во втором случае наличие постоянного соединения не нужно, но информацию о том, дошел ли пакет, или нет - получить невозможно (так называемые UDP или connectionless sockets). И первый и второй типы сокетов имеют своё практическое применение. Сокет необходимо создать как для сервера, так и для клиента.

Далее необходимо привязать адрес и порт к сокету. При помощи gethostbyname получаем локальный адрес компьютера. После этого заполняем структуру SOCKADDR_IN и вызываем функцию bind.

Затем с помощью функции sendto посылаем данные. Далее с помощью функции recvfrom принимаем данные. И в итоге перед

завершением программы как на сервере, так и на клиенте нужно закрыть сокет функцией closesocket. И, наконец, нужно очистить память, занимаемой библиотекой Windows Sockets.

Задание к работе

1. Разработать программу “Сервер” (на языке программирования Pascal или C), которая принимает запросы от клиентов и посылает им в качестве ответа некоторое сообщение.
2. Разработать программу “Клиент” (на языке программирования Pascal или C), которая посылает запрос серверу и “ждет” от него ответного сообщения.
3. Провести анализ функционирования разработанных программ (одновременная работа 2-х, 3-х и т.д. приложений на 2-х, 3-х и т.д. компьютерах ЛВС), сделать выводы.

Содержание отчета

1. Краткие теоретические сведения.
2. Используемые функции
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Контрольные вопросы

1. Сущность протокола IP.
2. Опишите структуру IP-адресов в классах A, B, C, D, E.
3. Какие IP-адреса являются зарезервированными для специального использования?
4. Что представляет собой маска подсети?
5. Как создать сокет для работы с протоколом IP?
6. Какие функции Winsocket необходимо вызвать для вывода на экран IP-адреса данного компьютера?
7. Разбейте сеть 10.10.0.0 / 15 на 8 частей. Запишите диапазоны доступных адресов в каждой из получившихся сетей.

Лабораторная работа № 4

Программирование протоколов TCP/UDP с использованием библиотеки Winsock

Цель работы: изучить протоколы TCP/UDP, основные функции библиотеки Winsock и составить программу для приема/передачи пакетов.

Краткие теоретические сведения

Протокол TCP

Transmission Control Protocol (TCP) (протокол управления передачей) - один из основных сетевых протоколов Интернета, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. Выполняет функции протокола транспортного уровня модели OSI.

TCP - это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета. В отличие от UDP гарантирует целостность передаваемых данных и уведомление отправителя о результатах передачи. Реализация TCP, как правило, встроена в ядро ОС, хотя есть и реализации TCP в контексте приложения.

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. Также TCP осуществляет надежную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере. Программы для электронной почты и обмена файлами используют TCP. TCP контролирует длину сообщения, скорость обмена сообщениями, сетевой трафик [12].

TCP протокол базируется на IP для доставки пакетов, но добавляются две важные вещи: во-первых устанавливается соединение - это позволяет ему, в отличие от IP, гарантировать доставку пакетов, во-вторых - используются порты для обмена пакетами между приложениями, а не просто узлами.

Протокол TCP предназначен для обмена данными — это «надежный» протокол, потому что он во-первых обеспечивает надежную доставку данных, так как предусматривает установления логического соединения; во-вторых, нумерует пакеты и подтверждает их прием квитанцией, а в случае потери организует повторную передачу; в-третьих, делит передаваемый поток байтов на части — сегменты - и передает их нижнему уровню, на приемной стороне снова собирает их в непрерывный поток байтов.

TCP соединение начинается с т.н. “рукопожатия”: узел A посылает узлу B специальный пакет SYN — приглашение к соединению; B отвечает пакетом SYN-ACK — согласием об установлении соединения; A посылает пакет ACK — подтверждение, что согласие получено.

После этого TCP соединение считается установленным, и приложения, работающие в этих узлах, могут посылать друг другу пакеты с данными.

«Соединение» означает, что узлы помнят друг о друге, нумеруют все пакеты, идущие в обе стороны, посылают подтверждения о получении каждого пакета и перепосылают потерявшиеся по дороге пакеты. Для узла A это соединение называется исходящим, а для узла B - входящим. Любое установленное TCP соединение симметрично, и пакеты с данными по нему всегда идут в обе стороны.

В отличие от традиционной альтернативы - UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения, которые должны быть созданы перед передачей данных. TCP соединение можно разделить на 3 стадии:

1. Установка соединения.
2. Передача данных.
3. Завершение соединения.

Протокол UDP

Протокол UDP (User Datagram Protocol) является одним из основных протоколов, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, немногим отличающиеся от услуг протокола IP. Протокол UDP обеспечивает доставку дейтограмм, но не требует подтверждения их получения. Протокол UDP не требует соединения с удаленным модулем UDP ("бессвязный" протокол). К заголовку IP-пакета UDP добавляет поля порт отправителя и порт получателя, которые обеспечивают мультиплексирование информации между различными прикладными процессами, а также поля длина UDP-дейтограммы и контрольная сумма, позволяющие поддерживать целостность данных. Таким образом, если на уровне IP для определения места доставки пакета используется адрес, на уровне UDP - номер порта [2, 4, 13, 14].

Протокол UDP ориентирован на транзакции, получение датаграмм и защита от дублирования не гарантированы. Приложения, требующие гарантированного получения потоков данных, должны использовать протокол управления пересылкой (Transmission Control Protocol - TCP).

UDP - минимальный ориентированный на обработку сообщений протокол транспортного уровня, задокументированный в RFC 768. UDP не предоставляет никаких гарантий доставки сообщения для протокола верхнего уровня и не сохраняет состояния отправленных сообщений.

UDP обеспечивает многоканальную передачу (с помощью номеров портов) и проверку целостности (с помощью контрольных сумм) заголовка и существенных данных. Надежная передача в случае необходимости должна реализовываться пользовательским приложением.

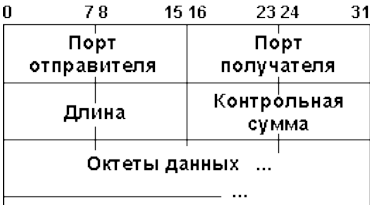

Рис. 4.1. Формат заголовка для датаграмм клиента

Заголовок UDP (рис. 4.1) состоит из четырех полей, каждое по 2 байта (16 бит). Два из них необязательны к использованию в IPv4, в то время как в IPv6 необязателен только порт отправителя.

В поле **Порт отправителя** указывается номер порта отправителя. Предполагается, что это значение задает порт, на который при необходимости будет посылаться ответ. В противном же случае, значение должно быть равным 0.

Поле **Порт получателя** обязательно и содержит порт получателя.

Поле **Длина датаграммы** задает длину всей датаграммы (заголовка и данных) в байтах. Минимальная длина равна длине заголовка – 8 байт. Теоретически, максимальный размер поля – 65535 байт для UDP-датаграммы (8 байт на заголовок и 65527 на данные). Фактический предел для длины данных при использовании IPv4 – 65507 (помимо 8 байт на UDP-заголовок требуется еще 20 на IP-заголовок).

Поле контрольной суммы используется для проверки заголовка и данных на ошибки. Если сумма не сгенерирована передатчиком, то поле заполняется нулями. Поле является обязательным для IPv6.

Из-за недостатка надежности, приложения UDP должны быть готовыми к некоторым потерям, ошибкам и дублированиям. Некоторые из них (например, TFTP) могут при необходимости добавить элементарные механизмы обеспечения надежности на прикладном уровне.

Но чаще такие механизмы не используются UDP-приложениями и даже мешают им. Потоковые медиа, многопользовательские игры в реальном времени и VoIP - примеры приложений, часто использующих протокол UDP. В этих конкретных приложениях

потеря пакетов обычно не является большой проблемой. Если приложению необходим высокий уровень надежности, то можно использовать другой протокол.

Более серьезной потенциальной проблемой является то, что в отличие от TCP, основанные на UDP приложения не обязательно имеют хорошие механизмы контроля и избежания перегрузок. Чувствительные к перегрузкам UDP-приложения, которые потребляют значительную часть доступной пропускной способности, могут поставить под угрозу стабильность в Интернете.

Функции библиотеки Winsock для работы с протоколами TCP/UDP

Инициализация Winsock, получение кодов ошибок, очистка памяти, создание и инициализация сокетов и т.д. происходит аналогично протоколу IP. Более подробно рассмотрим функции, необходимые для работы с TCP и UDP.

Перевод сокета в состояние “прослушивания” (для TCP) осуществляется функцией **listen (SOCKET s, int backlog),** где s – дескриптор сокета; backlog – максимальный размер очереди входящих сообщений на соединение. Используется сервером, чтобы информировать ОС, что он ожидает (“слушает”) запросы связи на данном сокете. Без этой функции всякое требование связи с сокетом будет отвергнуто.

Функция **connect (SOCKET s, const struct sockaddr FAR* name, int namelen)** нужна для соединения с сокетом, находящимся в состоянии “прослушивания” (для TCP). Она ипользуется процессом-клиентом для установления связи с сервером. В случае успешного установления соединения connect возвращает 0, иначе SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **accept (SOCKET s, struct sockaddr FAR* addr, int FAR* addrlen)** служит для подтверждения запроса на соединение (для TCP). Функция используется для принятия связи на сокет. Сокет

должен быть уже слушающим в момент вызова функции. Если сервер устанавливает связь с клиентом, то данная функция возвращает новый сокет-дескриптор, через который и производит общение клиента с сервером. Пока устанавливается связь клиента с сервером, функция блокирует другие запросы связи с данным сервером, а после установления связи “прослушивание” запросов возобновляется.

В случае автоматического распределения адресов и портов узнать какой адрес и порт присвоен сокету можно при помощи функции **getsockname (SOCKET s, struct sockaddr FAR* name, int FAR* namelen).** Если операция выполнена успешно, возвращает 0, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Для передачи данных по протоколу UDP используется функция **sendto (SOCKET s, const char FAR * buf, int len, int flags, const struct sockaddr FAR * to, int tolen).** Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Для передачи данных по протоколу TCP используется функция **send (SOCKET s, const char FAR * buf, int len, int flags),** где s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Для приема данных по протоколу UDP используется функция recvfrom **(SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen)**. Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Для приема данных по протоколу TCP используется функция **recv (SOCKET s, char FAR* buf, int len, int flags)**. Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Задание к работе

1. Разработать программу “Сервер” на языке программирования Pascal или C), которая принимает запросы от клиентов и посылает им в качестве ответа некоторое сообщение.
2. Разработать программу “Клиент” на языке программирования Pascal или C), которая посылает запрос серверу и “ждет” от него ответного сообщения.
3. Провести анализ функционирования разработанных программ (одновременная работа 2-х, 3-х и т.д. приложений на 2-х, 3-х и т.д. компьютерах ЛВС), сделать выводы.
4. Провести сравнительный анализ протоколов TCP и UDP. Сделать выводы.

Содержание отчета

1. Краткие теоретические сведения.
2. Используемые функции
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Контрольные вопросы

1. Что представляет собой протокол TCP? Как он работает?
2. Порядок установления TCP-соединения.
3. В чём состоит отличие протокола UDP от IP?
4. Формат заголовка пакета UDP.
5. Опишите работу функций sendto и send библиотеки Winsocket.
6. В каких случаях предпочтительней использовать протокол UDP?

Лабораторная работа № 5
Протоколы ARP/RARP

Цель работы: изучить протоколы ARP/RARP.

Краткие теоретические сведения

ARP (Address Resolution Protocol - протокол определения адреса) - протокол канального уровня, предназначенный для определения MAC-адреса (адреса канального уровня) по известному IP-адресу (адресу сетевого уровня). Наибольшее распространение этот протокол получил благодаря распространению сетей IP, построенных поверх Ethernet, поскольку практически в 100 % случаев при таком сочетании используется протокол ARP [16].

Протокол ARP работает различным образом в зависимости от того, какой протокол канального уровня работает в данной сети - протокол локальной сети (Ethernet, Token Ring, FDDI) с возможностью широковещательного доступа одновременно ко всем узлам сети, или же протокол глобальной сети (X.25, frame relay), как правило не поддерживающий широковещательный доступ.

Функциональность протокола ARP сводится к решению двух задач. Одна часть протокола определяет физические адреса при посылке дейтаграммы, другая отвечает на запросы устройств в сети. Протокол ARP предполагает, что каждое устройство «знает» как свой IP -адрес, так и свой физический адрес.

Для того чтобы уменьшить количество посылаемых запросов ARP, каждое устройство в сети, использующее протокол ARP , должно иметь специальную буферную память. В ней хранятся пары адресов (IP-адрес, физический адрес) устройств в сети. Всякий раз, когда устройство получает ARP-ответ, оно сохраняет в буферной памяти соответствующую пару. Если адрес есть в списке пар, то нет необходимости посылать ARP-запрос. Эта буферная память называется ARP-таблицей.

В ARP-таблице могут содержаться как статические, так и динамические записи. Динамические записи добавляются и удаляются автоматически, статические заносятся вручную. Так как большинство устройств в сети поддерживает динамическое разрешение адресов, то администратору, как правило, нет необходимости собственноручно указывать записи протокола ARP в таблице адресов.

Кроме того, ARP-таблица всегда содержит запись с физическим широковещательным адресом (OxFFFFFFFFFFFF) для локальной сети. Эта запись позволяет устройству принимать широковещательные ARP-запросы. Каждая запись в ARP-таблице имеет свое время жизни, например для операционной системы Microsoft Windows 2000 оно составляет 10 минут. При добавлении записи для нее активируется таймер. Если запись не востребована в первые две минуты, она удаляется. Если используется — будет существовать на протяжении 10 минут. В некоторых реализациях протокола ARP новый таймер устанавливается после каждого обращения к записи в ARP -таблице.

Сообщения протокола ARP при передаче по сети инкапсулируются в поле данных кадра. Они не содержат IP-заголовка. В отличие от сообщений большинства протоколов, сообщения ARP не имеют фиксированного формата заголовка. Это объясняется тем, что протокол был разработан таким образом, чтобы он был применим для разрешения адресов в различных сетях. Фактически протокол способен работать с произвольными физическими адресами и сетевыми протоколами.

Узел, которому нужно выполнить отображение IP-адреса на локальный адрес, формирует ARP запрос, вкладывает его в кадр протокола канального уровня, указывая в нем известный IP-адрес, и рассылает запрос широковещательно. Все узлы локальной сети получают ARP запрос и сравнивают указанный там IP-адрес с собственным. В случае их совпадения узел формирует ARP-ответ, в котором указывает свой IP-адрес и свой локальный адрес и отправляет его уже направленно, так как в ARP запросе отправитель указывает свой локальный адрес. ARP-запросы и ответы используют один и тот же формат пакета. Так как локальные адреса могут в различных типах

сетей иметь различную длину, то формат пакета протокола ARP зависит от типа сети. На рис. 5.1 показана структура запросов и ответов ARP и RARP.

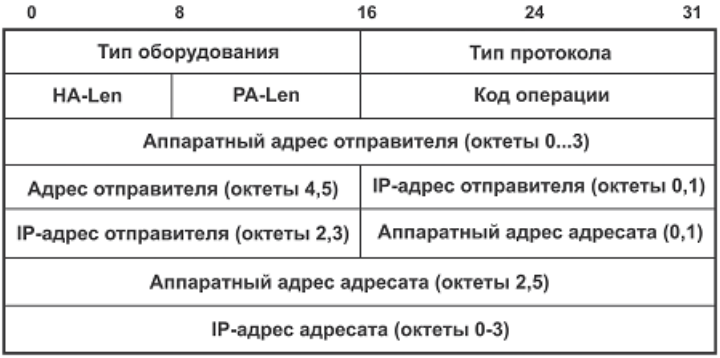

Рис.5.1. Структура запросов и ответов ARP и RARP

В поле типа оборудования для сетей Ethernet указывается значение 1. Поле типа протокола позволяет использовать пакеты ARP не только для протокола IP, но и для других сетевых протоколов. Для IP значение этого поля равно 0800 16. Длина локального адреса для протокола Ethernet равна 6 байтам, а длина IP-адреса - 4 байтам. В поле операции для ARP запросов указывается значение 1 для протокола ARP и 2 для протокола RARP.

Узел, отправляющий ARP-запрос, заполняет в пакете все поля, кроме поля искомого локального адреса (для RARP-запроса не указывается искомый IP-адрес). Значение этого поля заполняется узлом, опознавшим свой IP-адрес.

Рис. 5.2 и 5.3 показывают принцип работы протокола ARP. Анализируя структуру запроса, можно увидеть, что компьютер с адресом 192.168.3.2 делает попытку узнать MAC-адрес компьютера с IP -адресом 192.168.3.12. Для этого он посылает широковещательный запрос, содержащий IP-адрес, с MAC-адресом, установленным в FF:FF:FF:FF:FF:FF .

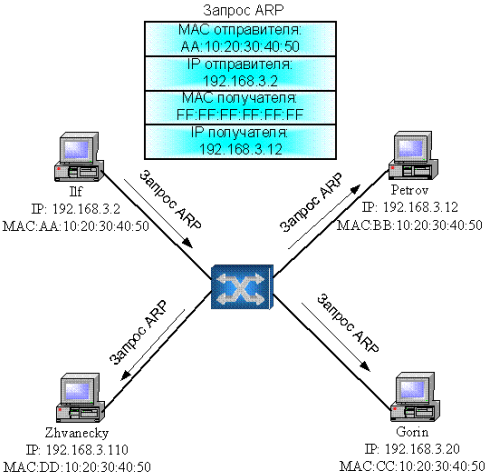

Рис. 5.2. Запрос протокола ARP

Когда компьютер с адресом 192.168.3.12 получает этот широковещательный запрос, он анализирует IP -адрес, для которого выполняется разрешение. Определив, что его адрес совпадает с искомым, он формирует ответ протокола ARP , где указывает свой MAC-адрес (рис. 5.3).

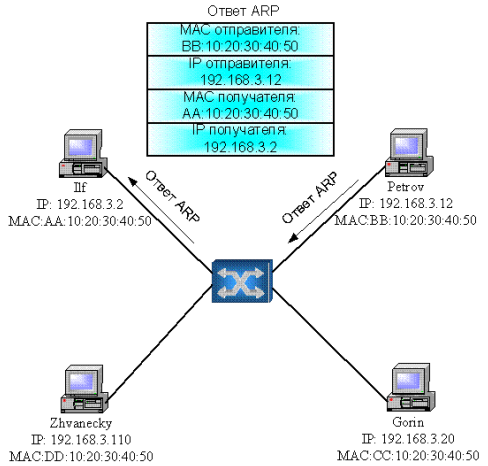

Рис. 5.3. Ответ протокола ARP

Ответ посылается уже не широковещательно - отправитель знает MAC-адрес инициатора запроса и поэтому передает пакет целенаправленно.

В глобальных сетях администратору сети чаще всего приходится вручную формировать ARP-таблицы, в которых он задает, например, соответствие IP-адреса адресу узла сети X.25, который имеет смысл локального адреса. В последнее время наметилась тенденция автоматизации работы протокола ARP и в глобальных сетях. Для этой цели среди всех маршрутизаторов, подключенных к какой-либо глобальной сети, выделяется специальный маршрутизатор, который ведет ARP-таблицу для всех остальных узлов и маршрутизаторов этой сети. При таком централизованном подходе для всех узлов и маршрутизаторов вручную нужно задать только IP-адрес и локальный адрес выделенного маршрутизатора. Затем каждый узел и маршрутизатор регистрирует свои адреса в выделенном маршрутизаторе, а при необходимости установления соответствия между IP-адресом и локальным адресом узел обращается к выделенному маршрутизатору с запросом и автоматически получает ответ без участия администратора.

Преобразование адресов выполняется путем поиска в таблице. Эта таблица, называемая ARP-таблицей, хранится в памяти и содержит строки для каждого узла сети. В двух столбцах содержатся IP- и Ethernet-адреса. Если требуется преобразовать IP-адрес в Ethernet-адрес, то ищется запись с соответствующим IP-адресом (табл. 5.1.).

Таблица 5.1.

Пример ARP-таблицы

IP-адрес	Ethernet-адрес
223.1.2.1	08:00:39:00:2F:C3
223.1.2.3	08:00:5A:21:A7:22
223.1.2.4	08:00:10:99:AC:54

Принято все байты 4-байтного IP-адреса записывать десятичными числами, разделенными точками. При записи 6-байтного Ethernet-

адреса каждый байт указывается в 16-ричной системе и отделяется двоеточием.

ARP-таблица необходима потому, что IP-адреса и Ethernet-адреса выбираются независимо, и нет какого-либо алгоритма для преобразования одного в другой. IP-адрес выбирает менеджер сети с учетом положения машины в сети internet. Если машину перемещают в другую часть сети internet, то ее IP-адрес должен быть изменен. Ethernet-адрес выбирает производитель сетевого интерфейсного оборудования из выделенного для него по лицензии адресного пространства. Когда у машины заменяется плата сетевого адаптера, то меняется и ее Ethernet-адрес.

В ходе обычной работы сетевая программа, например TELNET, отправляет прикладное сообщение, пользуясь транспортными услугами TCP. Модуль TCP посылает соответствующее транспортное сообщение через модуль IP. В результате составляется IP-пакет, который должен быть передан драйверу Ethernet. IP-адрес места назначения известен прикладной программе, модулю TCP и модулю IP. Необходимо на его основе найти Ethernet-адрес места назначения. Для определения искомого Ethernet-адреса используется ARP-таблица.

ARP-таблица заполняется автоматически модулем ARP, по мере необходимости. Когда с помощью существующей ARP-таблицы не удается преобразовать IP-адрес, то происходит следующее:

1. По сети передается широковещательный ARP-запрос.
2. Исходящий IP-пакет ставится в очередь.

Каждый сетевой адаптер принимает широковещательные передачи. Все драйверы Ethernet проверяют поле типа в принятом Ethernet-кадре и передают ARP-пакеты модулю ARP.

Каждый модуль ARP проверяет поле искомого IP-адреса в полученном ARP-пакете и, если адрес совпадает с его собственным IP-адресом, то посылает ответ прямо по Ethernet-адресу отправителя запроса.

Этот ответ получает машина, сделавшая ARP-запрос. Драйвер этой машины проверяет поле типа в Ethernet-кадре и передает ARP-

пакет модулю ARP. Модуль ARP анализирует ARP-пакет и добавляет запись в свою ARP-таблицу (см. табл. 5.2.).

Таблица 5.4.

ARP-таблица после обработки ответа

IP-адрес	Ethernet-адрес
223.1.2.1	08:00:39:00:2F:C3
223.1.2.2	08:00:28:00:38:A9
223.1.2.3	08:00:5A:21:A7:22
223.1.2.4	08:00:10:99:AC:54

Полностью порядок преобразования адресов выглядит так:

1. По сети передается широковещательный ARP-запрос.
2. Исходящий IP-пакет ставится в очередь.
3. Возвращается ARP-ответ, содержащий информацию о соответствии IP- и Ethernet-адресов. Эта информация заносится в ARP-таблицу.
4. Для преобразования IP-адреса в Ethernet-адрес у IP-пакета, постав ленного в очередь, используется ARP-таблица.
5. Ethernet-кадр передается по сети Ethernet.

Т.е., если с помощью ARP-таблицы не удается сразу осуществить преобразование адресов, то IP-пакет ставится в очередь, а необходимая для преобразования информация получается с помощью запросов и ответов протокола ARP, после чего IP-пакет передается по назначению.

Если в сети нет машины с искомым IP-адресом, то ARP-ответа не будет и не будет записи в ARP-таблице. Протокол IP будет уничтожать IP-пакеты, направляемые по этому адресу. Протоколы верхнего уровня не могут отличить случай повреждения сети Ethernet от случая отсутствия машины с искомым IP-адресом.

Некоторые реализации IP и ARP не ставят в очередь IP-пакеты на то время, пока они ждут ARP-ответов. Вместо этого IP-пакет просто уничтожается, а его восстановление возлагается на модуль TCP или прикладной процесс, работающий через UDP. Такое восстановление выполняется с помощью таймаутов и повторных передач. Повторная

передача сообщения проходит успешно, так как первая попытка уже вызвала заполнение ARP-таблицы.

Следует отметить, что каждая машина имеет отдельную ARP-таблицу для каждого своего сетевого интерфейса.

Протокол RARP - это протокол, решающий обратную задачу - нахождение IP-адреса по известному локальному адресу. Он называется реверсивный ARP - RARP (Reverse Address Resolution Protocol) и используется при старте бездисковых станций, не знающих в начальный момент своего IP-адреса, но знающих адрес своего сетевого адаптера. Reverse ARP (или обратное разрешение) работает аналогично протоколу ARP за исключением того, что в его задачи входит определение физического адреса по известному адресу сетевого уровня. Этот протокол требует наличия в сети сервера RARP , подключенного к тому же сегменту сети, что и интерфейс маршрутизатора. Наиболее часто протокол reverse ARP используется для запуска бездисковых рабочих станций [18].

В данной лабораторной работе совместно с библиотекой Windows Sockets необходимо использовать библиотеку функций IP Helper.

Задание

1. Программно реализовать вывод ARP-таблицы в следующем виде:

```
Интерфейс: 127.0.0.1 --- 0x1
  адрес в Интернете      Физический адрес      Тип
  224.0.0.2
статический

Интерфейс: 192.168.0.6 --- 0xb
  адрес в Интернете      Физический адрес      Тип
  192.168.0.4           00-1f-3a-66-98-fe
динамический
  192.168.0.100         00-00-00-00-00-00
недопустимый
  192.168.0.255         ff-ff-ff-ff-ff-ff
статический

Интерфейс: 0.0.0.0 --- 0xffffffff
```

```
  адрес в Интернете      Физический адрес      Тип
  224.0.0.2             01-00-5e-00-00-02
статический
```

2. Программно реализовать добавление записи в ARP-таблицу.
3. Программно реализовать удаление записи из ARP-таблицы.
4. Программно реализовать получение MAC-адреса по IP-адресу.
5. Программы должны быть написаны на языке программирования Pascal или C.

Содержание отчета

1. Краткие теоретические сведения.
2. Используемые функции
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Контрольные вопросы

1. Какие задачи решает протокол ARP?
2. Что такое ARP-таблица? Почему она является необходимым элементом?
3. Типы записей ARP-таблицы.
4. Опишите процесс преобразования ip-адреса в локальный.
5. Как может работать протокол ARP в глобальных сетях?
6. Что представляет собой протокол RARP?
7. В каких целях может быть использован протокол RARP?

Лабораторная работа № 6

Протоколы DHCP и DNS

Цель работы: изучить протоколы DHCP, DNS и составить программы согласно заданию.

Краткие теоретические сведения

Протокол DHCP

DHCP (англ. Dynamic Host Configuration Protocol — протокол динамической конфигурации узла) - это сетевой протокол, позволяющий компьютерам автоматически получать IP-адрес и другие параметры, необходимые для работы в сети TCP/IP. Данный протокол работает по модели «клиент-сервер». Для автоматической конфигурации компьютер-клиент на этапе конфигурации сетевого устройства обращается к так называемому серверу DHCP, и получает от него нужные параметры. Сетевой администратор может задать диапазон адресов, распределяемых сервером среди компьютеров. Это позволяет избежать ручной настройки компьютеров сети и уменьшает количество ошибок. Протокол DHCP используется в большинстве сетей TCP/IP.

Распределение IP-адресов

Протокол DHCP предоставляет три способа распределения IP-адресов:

1. *Ручное распределение.* При этом способе сетевой администратор сопоставляет аппаратному адресу (для Ethernet сетей это MAC-адрес) каждого клиентского компьютера определённый IP-адрес. Фактически, данный способ распределения адресов отличается от ручной настройки каждого компьютера лишь тем, что сведения об адресах хранятся централизованно (на сервере DHCP), и потому их проще изменять при необходимости.
2. *Автоматическое распределение.* При данном способе каждому компьютеру на постоянное использование выделяется произвольный свободный IP-адрес из определённого администратором диапазона.
3. *Динамическое распределение.* Этот способ аналогичен автоматическому распределению, за исключением того, что адрес выдаётся компьютеру не на постоянное пользование, а на определённый срок. Это называется *арендой адреса*. По истечении срока аренды IP-адрес вновь считается свободным, и клиент обязан запросить новый. Кроме того, клиент сам может отказаться от полученного адреса.

Некоторые реализации службы DHCP способны автоматически обновлять записи DNS, соответствующие клиентским компьютерам, при выделении им новых адресов. Это производится при помощи протокола обновления DNS, описанного в RFC 2136.

Помимо IP-адреса, DHCP также может сообщать клиенту дополнительные параметры, необходимые для нормальной работы в сети. Эти параметры называются опциями DHCP. Список стандартных опций можно найти в RFC 2132.

Некоторыми из наиболее часто используемых опций являются:

- IP-адрес маршрутизатора по умолчанию;
- маска подсети;
- адреса серверов DNS;
- имя домена DNS.

Протокол DHCP является клиент-серверным, то есть в его работе участвуют клиент DHCP и сервер DHCP. Передача данных производится при помощи протокола UDP, при этом сервер принимает сообщения от клиентов на порт 67 и отправляет сообщения клиентам на порт 68.

Все сообщения протокола DHCP разбиваются на поля, каждое из которых содержит определённую информацию. Все поля, кроме последнего (поля опций DHCP), имеют фиксированную длину.

Рассмотрим пример процесса получения IP-адреса клиентом от сервера DHCP. Предположим, клиент ещё не имеет собственного IP-адреса, но ему известен его предыдущий адрес — 192.168.1.100. Процесс состоит из четырёх этапов [21].

1. **Обнаружение DHCP.**

Вначале клиент выполняет широковещательный запрос по всей физической сети с целью обнаружить доступные DHCP-серверы. Он отправляет сообщение типа DHCPDISCOVER, при этом в качестве IP-адреса источника указывается 0.0.0.0 (так как компьютер ещё не имеет собственного IP-адреса), а в качестве адреса назначения - широковещательный адрес 255.255.255.255.

Клиент заполняет несколько полей сообщения начальными значениями:

- В поле **xid** помещается уникальный *идентификатор транзакции*, который позволяет отличать данный процесс получения IP-адреса от других, протекающих в то же время.
- В поле **chaddr** помещается аппаратный адрес (MAC-адрес) клиента.
- В поле опций указывается последний известный клиенту IP-адрес. В данном примере это 192.168.1.100. Это необязательно и может быть проигнорировано сервером.

Сообщение DHCPDISCOVER может быть распространено за пределы локальной физической сети при помощи специально настроенных агентов ретрансляции DHCP, перенаправляющих поступающие от клиентов сообщения DHCP серверам в других подсетях.

2. **Предложение DHCP.**

Получив сообщение от клиента, сервер определяет требуемую конфигурацию клиента в соответствии с указанными сетевым администратором настройками. Пусть в данном случае DHCP-сервер согласен с запрошенным клиентом адресом 192.168.1.100. Сервер отправляет ему ответ (DHCPOFFER), в котором предлагает конфигурацию. Предлагаемый клиенту IP-адрес указывается в поле yiaddr. Прочие параметры (такие, как адреса маршрутизаторов и DNS-серверов) указываются в виде опций в соответствующем поле.

Далее это сообщение DHCP-сервер отправляет хосту, пославшему DHCPDISCOVER, на его MAC, при определенных обстоятельствах сообщение может распространяться как широковещательная рассылка. Клиент может получить несколько различных предложений DHCP от разных серверов; из них он должен выбрать то, которое его «устраивает».

3. **Запрос DHCP.**

Выбрав одну из конфигураций, предложенных DHCP-серверами, клиент отправляет запрос DHCP (DHCPREQUEST). Он рассылается широковещательно; при этом к опциям, указанным клиентом в сообщении DHCPDISCOVER, добавляется специальная опция -

идентификатор сервера - указывающая адрес DHCP-сервера, выбранного клиентом (в данном случае - 192.168.1.1).

4. **Подтверждение DHCP.**

Наконец, сервер подтверждает запрос и направляет это подтверждение (DHCPACK) клиенту. После этого клиент должен настроить свой сетевой интерфейс, используя предоставленные опции.

Протокол DNS

В стеке TCP/IP применяется доменная система имен, которая имеет иерархическую древовидную структуру.

Иерархия доменных имен аналогична иерархии имен файлов, принятой в файловых системах. В отличие от имен файлов запись доменного имени начинается с самой младшей составляющей, и заканчивается самой старшей. Составные части доменного имени отделяются друг от друга точкой.

Разделение имени на части позволяет разделить административную ответственность за назначение уникальных имен между различными людьми или организациями в пределах своего уровня иерархии. Разделение административной ответственности позволяет решить проблему образования уникальных имен без взаимных консультаций между организациями, отвечающими за имена одного уровня иерархии. Поэтому должна существовать одна организация, отвечающая за назначение имен верхнего уровня иерархии.

Совокупность имен, у которых несколько старших составных частей совпадают, образуют домен (domain) имен.

Если один домен входит в другой домен как его составная часть, то такой домен могут называть поддоменом (subdomain). Обычно поддомен называют по имени той его старшей составляющей, которая отличает его от других поддоменов. Имя поддомену назначает администратор вышестоящего домена. Хорошей аналогией домена является каталог файловой системы.

По аналогии с файловой системой в доменной системе имен различают краткие имена, относительные имена и полные доменные

имена. Краткое имя - это имя конечного узла сети. Относительное имя - это составное имя, начинающееся с некоторого уровня иерархии, но не самого верхнего. Например, wwwl.zil — это относительное имя. Полное доменное имя (fully qualified domain name, FQDN) включает составляющие всех уровней иерархии, начиная от краткого имени и кончая корневой точкой.

Корневой домен управляется центральными органами Интернета: IANA и InterNIC. Домены верхнего уровня назначаются для каждой страны, а также на организационной основе. Имена этих доменов должны следовать международному стандарту ISO 3166. Для обозначения стран используются трехбуквенные и двухбуквенные аббревиатуры, например, ru (Россия), uk (Великобритания), fin (Финляндия), us (Соединенные Штаты), а для различных типов организаций - следующие обозначения: com - коммерческие организации; edu - образовательные организации; gov - правительственные организации; org - некоммерческие организации; net - организации поддержки сетей.

Каждый домен администрируется отдельной организацией, которая обычно разбивает свой домен на поддомены и передает функции администрирования этих поддоменов другим организациям. Чтобы получить доменное имя, необходимо зарегистрироваться в какой-либо организации, которой организация InterNIC делегировала свои полномочия по распределению имен доменов. В России такой организацией является РосНИИРОС, которая отвечает за делегирование имен поддоменов в домене ru.

Доменная система имен реализована в Интернете, но она может работать и как автономная система имен в любой крупной корпоративной сети, которая также использует стек TCP/IP, но не связана с Интернетом.

Соответствие между доменными именами и IP-адресами может устанавливаться как средствами локального хоста, так и средствами централизованной службы. На раннем этапе развития Интернета на каждом хосте вручную создавался текстовый файл с известным

именем hosts.txt. Этот файл состоял из некоторого количества строк, каждая из которых содержала одну пару «IP-адрес — доменное имя».

В настоящее время используется масштабируемая служба для разрешения имен — система доменных имен (Domain Name System, DNS). Служба DNS использует в своей работе протокол типа «клиент-сервер». В нем определены DNS-серверы и DNS-клиенты. DNS-серверы поддерживают распределенную базу отображений, а DNS-клиенты обращаются к серверам с запросами о разрешении доменного имени в IP-адрес.

Служба DNS использует текстовые файлы, которые администратор подготавливает вручную. Однако служба DNS опирается на иерархию доменов, и каждый сервер службы DNS хранит только часть имен сети, а не все имена. При росте количества узлов в сети проблема масштабирования решается созданием новых доменов и поддоменов имен и добавлением в службу DNS новых серверов.

Для каждого домена имен создается свой DNS-сервер. Обычно сервер домена хранит только имена, которые заканчиваются на следующем ниже уровне иерархии по сравнению с именем домена. Именно при такой организации службы DNS нагрузка по разрешению имен распределяется более-менее равномерно между всеми DNS-серверами сети. Каждый DNS-сервер кроме таблицы отображений имен содержит ссылки на DNS-серверы своих поддоменов. Эти ссылки связывают отдельные DNS-серверы в единую службу DNS. Ссылки представляют собой IP-адреса соответствующих серверов. Для обслуживания корневого домена выделено несколько дублирующих друг друга DNS-серверов, IP-адреса которых являются широко известными (их можно узнать в InterNIC).

Процедура разрешения DNS-имени во многом аналогична процедуре поиска файловой системой адреса файла по его символьному имени. Для определения IP-адреса по доменному имени необходимо просмотреть все DNS-серверы, обслуживающие цепочку поддоменов, входящих в имя хоста, начиная с корневого домена. Существенным отличием является то, что файловая система

расположена на одном компьютере, а служба DNS по своей природе является распределенной.

Существует две основные схемы разрешения DNS-имен. В первом варианте работу по поиску IP-адреса координирует DNS-клиент, последовательно обращаясь к DNS-серверам, начиная с корневого. Такая схема взаимодействия называется нерекурсивной, или итеративной. Так как эта схема загружает клиента достаточно сложной работой, то она применяется редко. Во втором варианте DNS-клиент запрашивает локальный DNS-сервер, обслуживающий поддомен, к которому принадлежит имя клиента. Если локальный DNS-сервер знает ответ, то он сразу же возвращает его клиенту. Это может соответствовать случаю, когда запрошенное имя входит в тот же поддомен, что и имя клиента, а также случаю, когда сервер уже узнавал данное соответствие для другого клиента и сохранил его в своем кэше. Если локальный сервер не знает ответ, то он выполняет итеративные запросы к корневому серверу и к нижним по иерархии. Получив ответ, он передает его клиенту. Такая схема называется косвенной, или рекурсивной.

Для ускорения поиска IP-адресов DNS-серверы широко применяют процедуру кэширования проходящих через них ответов. Чтобы служба DNS могла оперативно отрабатывать изменения, происходящие в сети, ответы кэшируются на определенное время — обычно от нескольких часов до нескольких дней.

В данной лабораторной работе совместно с библиотекой Windows Sockets необходимо использовать библиотеку функций IP Helper. Подробная информация по её подключению находится в разделе Getting Started With IP Helper на MSDN [19].

Следует учитывать, что функции DhcpCApiInitialize и DhcpCApiCleanup, могут отсутствовать в библиотеке IP Helper. Это зависит от языка и среды программирования, а также версии данной библиотеки. В таком случае инициализация ресурсов DHCP не нужна; другие функции остаются работоспособными.

Задание

1. Разработать программу, позволяющую по доменному имени узнать IP-адрес узла, а по IP-адресу узнать доменное имя узла.
2. Разработать программу, которая выводит информацию о DHCP–сервере и позволяет сбросить и запросить IP-адрес у этого сервера.
3. Программы должны быть написаны на языке программирования Pascal или C.

Содержание отчета

1. Краткие теоретические сведения.
2. Используемые функции.
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Контрольные вопросы

1. Что представляет собой протокол DHCP?
2. Способы распределения IP-адресов.
3. Опишите процесс получения клиентом IP-адреса от DHCP-сервера.
4. В чём состоит суть доменной системы имён?
5. Типы доменных имён.
6. Как происходит управление доменами?
7. Состав службы DNS.
8. Опишите процесс разрешения доменного имени по двум существующим схемам.

Лабораторная работа № 7

Протоколы POP3 и SMTP

Цель работы: изучить принципы и характеристику протоколов POP3 и SMTP и составить программу для приема/отправки электронной почты.

Краткие теоретические сведения

Протокол POP3

POP3 (англ. Post Office Protocol Version 3) - стандартный Интернет-протокол прикладного уровня, используемый клиентами электронной почты для извлечения электронного сообщения с удаленного сервера по TCP/IP-соединению [22].

В некоторых небольших узлах Интернет бывает непрактично поддерживать систему передачи сообщений (MTS - Message Transport System). Рабочая станция может не иметь достаточных ресурсов для обеспечения непрерывной работы SMTP-сервера [RFC-821]. Для “домашних ЭВМ” слишком дорого поддерживать связь с Интернет круглые сутки.

Но доступ к электронной почте необходим как для таких малых узлов, так и индивидуальных ЭВМ. Для решения этой проблемы разработан протокол POP3 (Post Office Protocol - Version 3, STD- 53. M. Rose, RFC-1939). Этот протокол обеспечивает доступ узла к базовому почтовому серверу.

POP3 не ставит целью предоставление широкого списка манипуляций с почтой. Почтовые сообщения принимаются почтовым сервером и сохраняются там, пока на рабочей станции клиента не будет запущено приложение POP3. Это приложение устанавливает соединение с сервером и забирает сообщения оттуда. Почтовые сообщения на сервере стираются.

POP3 поддерживает простые требования «загрузи-и-удали» для доступа к удаленным почтовым ящикам. Хотя большая часть POP-клиентов предоставляют возможность оставить почту на сервере после загрузки, использующие POP клиенты обычно соединяются, извлекают все письма, сохраняют их на пользовательском компьютере

как новые сообщения, удаляют их с сервера, после чего разъединяются.

Другие протоколы, в частности IMAP, предоставляют более полный и комплексный удаленный доступ к типичным операциям с почтовым ящиком. Многие клиенты электронной почты поддерживают как POP, так и IMAP; однако, гораздо меньше интернет-провайдеров поддерживают IMAP.

POP3-сервер прослушивает порт 110. Шифрование связи для POP3 запрашивается после запуска протокола, с помощью либо команды STLS (если она поддерживается), либо POP3S, которая соединяется с сервером используя TLS или SSL по TCP-порту 995.

Доступные сообщения клиента фиксируются при открытии почтового ящика POP-сессией и определяются количеством сообщений для сессии, или, по желанию, с помощью уникального идентификатора, присваиваемого сообщению POP-сервером. Этот уникальный идентификатор является постоянным и уникальным для почтового ящика и позволяет клиенту получить доступ к одному и тому же сообщению в разных POP-сессиях. Почта извлекается и помечается для удаления с помощью номера сообщения. При выходе клиента из сессии помеченные сообщения удаляются из почтового ящика.

Обычно POP3-сервис устанавливается на 110-й TCP -порт сервера, который будет находиться в режиме ожидания входящего соединения. Когда клиент хочет воспользоваться POP3-сервисом, он просто устанавливает TCP-соединение с портом 110 этого хоста. После установления соединения сервис POP3 отправляет подсоединившемуся клиенту приветственное сообщение. После этого клиент и сервер начинают обмен командами и данными. По окончании обмена POP3-канал закрывается.

Команды POP3 состоят из ключевых слов, состоящих из ASCII-символов, и одним или несколькими параметрами, отделяемыми друг от друга символом "пробела" - <SP>. Все команды заканчиваются символами "возврата каретки" и "перевода строки" - <CRLF>. Длина

ключевых слов не превышает четырех символов, а каждого из аргументов может быть до 40 символов.

Ответы POP3-сервера на команды состоят из строки статус-индикатора, ключевого слова, строки дополнительной информации и символов завершения строки. Длина строки ответа может достигать 512 символов. Строка статус-индикатора принимает два значения: положительное ("+OK") и отрицательное ("-ERR"). Любой сервер POP3 обязан отправлять строки статус-индикатора в верхнем регистре, тогда как другие команды и данные могут приниматься или отправляться как в нижнем, так и в верхнем регистрах.

Ответы POP3-сервера на отдельные команды могут составлять несколько строк. В этом случае строки разделены символами <CRLF>. Последнюю строку информационной группы завершает строка, состоящая из символа "." (код — 046) и <CRLF>, т. е. последовательность "CRLF.CRLF".

POP3-сессия состоит из нескольких частей. Как только открывается TCP-соединение и POP3-сервер отправляет приветствие, сессия должна быть зарегистрирована - состояние аутентификации (AUTHORIZATION state). Клиент должен зарегистрироваться в POP3-сервере, т. е. ввести свой идентификатор и пароль. После этого сервер предоставляет клиенту его почтовый ящик и открывает для данного клиента транзакцию - состояние начала транзакции обмена (TRANSACTION state). На этой стадии клиент может считать и удалить почту своего почтового ящика.

После того как клиент заканчивает работу (передает команду QUIT), сессия переходит в состояние UPDATE - завершение транзакции. В этом состоянии POP3-сервер закрывает транзакцию данного клиента (на языке баз данных - операция COMMIT) и закрывает TCP-соединение. В случае получения неизвестной, неиспользуемой или неправильной команды, POP3-сервер должен ответить отрицательным состоянием индикатора.

POP3-сервер может использовать в своей работе таймер контроля времени соединения. Этот таймер отсчитывает время "бездействия" ("idle") клиента в сессии от последней переданной команды. Если

время сессии истекло, сервер закрывает TCP-соединение, не переходя в состояние UPDATE (иными словами, откатывает транзакцию или на языке баз данных — выполняет ROLLBACK) [25]. POP3-сервер может обслуживать группу клиентов, которые, возможно, присоединяются по коммутируемой линии, и, следовательно, необходимо иметь средство автоматического регулирования времени соединения. По спецификации POP3-таймер контроля состояния "idle" должен быть установлен на промежуток времени не менее 10 минут.

Команды протокола POP3:

- 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;
- RETR - извлекает сообщение из почтового ящика;
- DELE - отмечает сообщение для удаления;
- 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.

Протокол SMTP

SMTP (Simple Mail Transfer Protocol) - широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP. SMTP впервые был описан в RFC 821 (1982 год); последнее обновление в RFC 5321 (2008) включает масштабируемое расширение - ESMTP (Extended SMTP). В настоящее время под «протоколом SMTP», как правило, подразумевают и его расширения. Протокол SMTP предназначен для передачи исходящей почты, используя для этого порт TCP 25 [23].

Упрощенно схема взаимодействия представлена на рис. 7.1 (объемными стрелками показано направление движения почтовых сообщений).

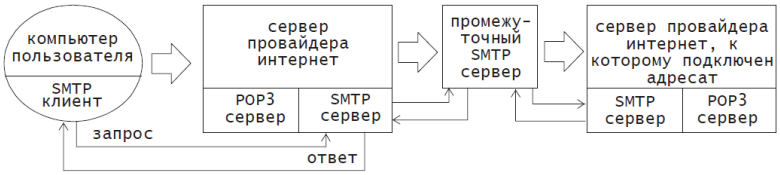

Рис. 7.1. Отправка почты по протоколу SMTP

Со стороны пользователя обычно одна и та же программа выступает в роли и POP3 клиента, и SMTP клиента отправителя. Наиболее распространенными на данный момент являются MS Outlook, The Bat, Netscape Messenger, Eudora, Pegasus mail, Mutt, Pine и др. При нажатии в них на кнопочку "отправить" происходит формирование очереди сообщений, и установление двустороннего сеанса общения с SMTP сервером провайдера. На схеме у пользователя есть клиентское ПО, а у провайдера – серверная часть приложения. На самом деле это немного не так. Протокол SMTP делает возможным смену сторон даже в ходе одного сеанса. Условно принято считать клиентом ту сторону, которая начинает взаимодействие и хочет отослать почту, а сервером ту, что принимает запросы. После того, как клиент посылает серверу несколько служебных команд и получает положительные ответы на них, он отправляет SMTP серверу собственно тело сообщения. SMTP сервер получает сообщение, вносит в него дополнительные заголовки, указывающие на то, что он обработал данное послание, устанавливает связь со следующим SMTP сервером по пути следования письма. Общение между любыми SMTP серверами происходит по той же схеме. Инициирует переговоры клиент, сервер на них отвечает, а затем получает корреспонденцию и "ставит штампик" в теле письма (в его заголовочной части). Все это очень напоминает обычную бумажную почту, где работу по сортировке и отправке почты выполняют люди.

Если на каком-нибудь этапе передачи SMTP клиент обнаружит невозможность подключиться к следующему серверу (например,

компьютер отправили на профилактику или аппаратура связи вышла из строя), он будет пытаться отправить сообщение через некоторое время – 1 час, 4 часа, день и т.д. до 4 суток в общем случае. Причем временные отрезки между попытками, как правило, зависят от настроек программы-пересыльщика почты. Одновременно, такой сервер должен уведомить отправителя сообщения о невозможности доставить почту, послав ему стандартное письмо "Failed delivery" (доставка невозможна) и рассказав о графике дальнейших попыток по продвижению исходного сообщения. Если канал связи не восстановится за указанный большой промежуток времени (например, 4 дня), посланная информация будет считаться утерянной.

Как только почта достигнет конечного пункта (SMTP сервера адресата сообщения), она будет сложена в почтовый ящик абонента, который всегда сможет в удобное для него время изъять ее по протоколам POP3 или IMAP, в зависимости от того, какой из них поддерживается провайдером.

Анализируя заголовок письма, можно узнать какими путями оно путешествовало, как долго длился сам путь, как называлась почтовая программа отправителя и многое другое. Получить эту информацию можно в "Свойствах письма", кликнув правой кнопкой мыши на самом письме в MS Outlook, нажав Ctrl+Shift+H в The Bat или совершить нечто подобное в других почтовых клиентах.

Рассмотрим клиент-серверное взаимодействие по протоколу SMTP. Программа пользователя, выбрав для связи соответствующий почтовый сервер, устанавливает с ним контакт на транспортном и сеансовом уровнях эталонной модели взаимодействия открытых систем OSI/RM (в терминах TCP/IP (transmission control protocol / internet protocol) это – TCP уровень). Взаимодействие на более низких уровнях (канальном, сетевом) происходит прозрачно для обеих сторон. Протокол SMTP – протокол прикладного уровня и базируется поверх TCP. В его рамках не оговаривается ни размер сегментов данных, ни правила квитирования, ни отслеживание ошибок, возникающих при передаче информации.

По уже установленному соединению клиентское ПО передает команды SMTP серверу, ожидая тут же получить ответы. В арсенал SMTP клиента, равно как и сервера, входит около 10 команд, но, воспользовавшись только пятью из них, уже можно легально послать почтовое сообщение. Это HELO, MAIL, RCPT, DATA, QUIT. Их использование подразумевается именно в такой последовательности. HELO предназначена для идентификации отправителя, MAIL указывает адрес отправителя, RCPT – адрес назначения. После команды DATA и ответа на нее, клиент посылает серверу тело сообщения, которое должно заканчиваться строкой, содержащей лишь одну точку.

Непосредственно после установления соединения сервер выдает строчку с кодом ответа 220. В ответ на нее клиент может инициировать сеанс связи по протоколу SMTP, послав команду HELO и указав у нее в аргументах имя своего компьютера. По принятии команды HELO сервер обязан сделать запрос в DNS и, если это возможно, по IP адресу определить доменное имя компьютера клиента. (IP адрес уже известен на момент установления соединения по TCP протоколу).

Далее в команде "MAIL FROM:" клиент сообщает обратный адрес отправителя, который проверяется обычно только на корректность. После слов "RCPT TO:" следует набрать адрес электронной почты абонента на данном сервере. Клиент отсылает команду DATA и ждет приглашения начать пересылку тела письма (код 354).

Сообщение может быть достаточно длинным, но обязательно должно заканчиваться строкой, в которой есть одна-единственная точка. Это служит сигналом SMTP серверу о том, что тело письма закончилось. Он присваивает этому письму определенный идентификатор, и ждет команды QUIT, после чего сеанс считается завершенным.

Если клиент посылает сообщение, у которого в заголовочной части в поле CC указаны несколько e-mail адресов, первый по пути следования SMTP сервер должен будет в общем случае установить сеанс продвижения почты с каждым из серверов данного списка и

отослать точную копию письма каждому. В случае использования поля BCC клиент, формирующий сообщение, уничтожит запись BCC в теле сообщения и по количеству адресатов отошлет первому SMTP серверу команду "RCPT TO:" каждый раз с новым адресом в качестве аргумента. Таким образом, сервер получит указание разослать почту по многим адресатам. Причем, в этом случае получатели писем ничего не будут знать друг о друге, т.к. рассылка осуществляется посредством команд SMTP протокола.

Задание к работе

1. Разработать программу, которая отправляет письмо с вложениями на указанный электронный адрес.
2. Разработать программу, получающую ответ, отправленный на указанный электронный адрес.
3. Программы должны быть написаны на языке программирования Pascal или C.

Для выполнения практической части данной лабораторной работы достаточно использовать ранее изученные функции библиотеки WinSock.

В программе для отправки письма пользователь должен указать почтовый сервер, почтовый адрес отправителя и получателя, логин, пароль, тему письма, текст сообщения, отправляемый файл. По окончании передачи письма необходимо вывести лог сессии.

В программе для получения списка писем пользователь должен указать почтовый сервер, логин и пароль. Сообщения выводятся в виде списка с номером и размером. Пользователь может прочитать или удалить любое из них.

Содержание отчета

1. Краткие теоретические сведения.
2. Используемые функции
3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Контрольные вопросы

1. Что представляет собой протокол POP3? С какой целью он был разработан?
2. Опишите процесс работы протокола POP3.
3. Формат команд протокола POP3.
4. Из каких частей состоит POP3-сессия?
5. Как осуществляется взаимодействие SMTP и POP3?
6. Минимальный набор команд и порядок их применения для отправки почты по протоколу SMTP.

Лабораторная работа № 8

Программирование протокола HTTP

Цель работы: изучить протокол HTTP и составить программу согласно заданию.

Краткие теоретические сведения

HTTP (Hyper Text Transfer Protocol – протокол передачи гипертекста) – протокол прикладного уровня стека протоколов TCP/IP, предназначенный для передачи данных по сети с использованием транспортного протокола TCP. Текущая версия протокола HTTP v1.1, его спецификация приводится в документе RFC 2616.

Протокол HTTP может использоваться также в качестве «транспорта» для других протоколов прикладного уровня, таких как SOAP или XML-RPC.

Основой HTTP является технология «клиент-сервер». HTTP-клиенты отсылают HTTP-запросы, которые содержат метод, обозначающий потребность клиента. Также такие запросы содержат универсальный идентификатор ресурса, указывающий на желаемый ресурс. Обычно такими ресурсами являются хранящиеся на сервере файлы. По умолчанию HTTP-запросы передаются на порт 80. HTTP-сервер отсылает коды состояния, сообщая, успешно ли выполнен HTTP-запрос или же нет.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента.

Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т.д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

Унифицированный идентификатор ресурса представляет собой сочетание унифицированного указателя ресурса (Uniform Resource Locator, URL) и унифицированного имени ресурса (Uniform Resource Name, URN). Например:

URI= http://iitus.bstu.ru/to_schoolleaver/230400

URL= http://iitus.bstu.ru/

URN= to_schoolleaver/230400

Метод протокола HTTP – это команда, передаваемая HTTP-клиентом HTTP-серверу. В табл. 8.1. перечислены некоторые методы, определенные в протоколе HTTP v1.1. Полный список методов HTTP v1.1. содержится в документе RFC 2616.

Таблица 8.1

Основные методы HTTP v1.1

Метод	Назначение
GET	Используется для запроса содержимого ресурса, на который указывает URI, содержащийся в запросе
HEAD	Используется для извлечения метаданных или проверки наличия ресурса, на который указывает URI, содержащийся в запросе
POST	Применяется для передачи данных заданному ресурсу. Данный метод предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого
PUT	Применяется для передачи данных заданному ресурсу. Данный метод предполагает, что передаваемое клиентом содержимое соответствует находящемуся по данному URI ресурсу

OPTIONS	Используется для определения возможностей HTTP-сервера или параметров соединения для конкретного ресурса
DELETE	Применяется для удаления ресурса, на который указывает URI

Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Название метода чувствительно к регистру. Каждый сервер обязан поддерживать как минимум методы GET и HEAD. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (NotImplemented). Если серверу метод известен, но он неприменим к конкретному ресурсу, то возвращается сообщение с кодом 405 (MethodNotAllowed). В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

Код состояния HTTP представляет собой целое число из трех цифр. Первая цифра указывает на класс состояния:

- информационные сообщения;
- успешное выполнение;
- переадресация;
- ошибка клиента;
- ошибка сервера.

Полный список статусов HTTP v1.1. содержится в документе RFC 2616. Примеры:

201 Webpage Created

403 Access allowed only for registered users

Каждое HTTP-сообщение состоит из трех частей, которые передаются в следующем порядке:

1. Стартовая строка – определяет тип сообщения;

2. Заголовки – характеризуют тело сообщения, параметры передачи и прочие сведения;

3. Тело сообщения – непосредственно данные сообщения.

Стартовые строки HTTP-сообщения различаются для запроса и ответа. Стартовая строка HTTP-запроса имеет следующий формат:

Метод URI HTTP/Версия, где

метод - название запроса,

URI определяет путь к запрашиваемому документу,

версия - пара разделённых точкой арабских цифр.

Стартовая строка HTTP-ответа имеет следующий формат:

HTTP/Версия КодСостояния Пояснение.

Заголовок HTTP представляет собой строку в HTTP-сообщении, содержащую разделенную двоеточием пару параметр-значение. Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой. Все заголовки разделяются на четыре основных группы:

1. Основные заголовки (General Headers) – должны включаться в любое сообщение клиента и сервера.
2. Заголовки запроса (Request Headers) – используются только в запросах клиента.
3. Заголовки ответа (Response Headers) – только для ответов от сервера.
4. Заголовки сущности (Entity Headers) – сопровождают каждую сущность сообщения.

Полный список заголовков HTTP v1.1. содержится в документе RFC 2616. Примеры заголовков:

Content-Type: text/plain; charset=windows-1251

Content-Language: ru

Тело HTTP-сообщения, если оно присутствует, используется для передачи данных, связанных с запросом или ответом.

Чтобы понять, как работает протокол HTTP, рассмотрим пример получения HTML-страницы с HTTP-сервера.

HTTP-запрос:

GET /to_schoolleaver/230400 HTTP/1.1
Host: iitus.bstu.ru
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)
Accept: text/html
Accept-Language: ru
Connection: close

HTTP-ответ:

HTTP/1.1 200 OK
Date: Fri, 16 Dec 2012 13:45:00 GMT
Server: Apache
Last-Modified: Wed, 11 Feb 2009 11:20:59 GMT
Content-Language: ru
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Connection: close
(далее следует запрошенная HTML-страница)

Протокол HTML позволяет достаточно легко создавать клиентские приложения. Возможности протокола можно расширить благодаря внедрению своих собственных заголовков, с помощью которых можно получить необходимую функциональность при решении нетривиальной задачи. При этом сохраняется совместимость с другими клиентами и серверами: они будут просто игнорировать неизвестные им заголовки.

Протокол HTTP устанавливает отдельную TCP-сессию на каждый запрос; в более поздних версиях HTTP было разрешено делать несколько запросов в ходе одной TCP-сессии, но браузеры обычно запрашивают только страницу и включённые в неё объекты (картинки, каскадные стили и т. п.), а затем сразу разрывают TCP-сессию. Для поддержки авторизованного (неанонимного) доступа в HTTP используются cookies; причём такой способ авторизации позволяет сохранить сессию даже после перезагрузки клиента и сервера.

При доступе к данным по FTP или по файловым протоколам тип файла (точнее, тип содержащихся в нём данных) определяется по расширению имени файла, что не всегда удобно. HTTP перед тем, как передать сами данные, передаёт заголовок «Content-Type: тип/подтип», позволяющую клиенту однозначно определить, каким образом обрабатывать присланные данные. Это особенно важно при работе с CGI-скриптами, когда расширение имени файла указывает не на тип присылаемых клиенту данных, а на необходимость запуска данного

файла на сервере и отправки клиенту результатов работы программы, записанной в этом файле (при этом один и тот же файл в зависимости от аргументов запроса и своих собственных соображений может порождать ответы разных типов — в простейшем случае картинки в разных форматах).

Кроме того, HTTP позволяет клиенту прислать на сервер параметры, которые будут переданы запускаемому CGI-скрипту. Для этого же в HTML были введены формы.

Перечисленные особенности HTTP позволили создавать поисковые машины (первой из которых стала AltaVista, созданная фирмой DEC), форумы и Internet-магазины. Это коммерциализировало Интернет, появились компании, основным полем деятельности которых стало предоставление доступа в Интернет (провайдеры) и создание сайтов.

В данной лабораторной работе необходимо использовать функции работы с Winsock из лабораторной работы №4. Подробное их описание можно найти на MSDN.

Задание к работе

1. Разработать программу, позволяющую принимать запрос на выдачу страницы от интернет-браузера и формировать ответ в зависимости от запроса. Реализовать методы GET и HEAD на стороне сервера.
2. Передать браузеру в сообщении тестовую страницу (определяется программистом) запрошенную страницу, или код ошибки если страница не найдена.
3. Программы должны быть написаны на языке программирования Pascal или C.

Содержание отчета

1. Краткие теоретические сведения.
2. Используемые функции

3. Разработка программы. Блок-схемы программы.
4. Анализ функционирования разработанных программ.
5. Выводы.
6. Тексты программ. Скриншоты программ.

Контрольные вопросы

1. Как расшифровывается аббревиатура HTTP?
2. Какой уровень занимает протокол в стеке TCP/IP?
3. На какой технологии построен протокол HTTP?
4. Какие преимущества протокола HTTP?
5. Какие недостатки протокола HTTP?
6. Какие методы существуют в протоколе HTTP?
7. Какие нововведения содержит версия HTTP 1.1?
8. Какова структура протокола HTTP? Охарактеризуйте каждый элемент
9. Какие существуют классы кодов состояния?
10. Какие существуют группы заголовков HTTP?
11. Что такое cookie-файлы? Для чего они используются?
12. Что такое HTTP referrer? Для чего он используется?

Библиографический список

1. «Википедия» — свободная энциклопедия. IPX [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/IPX – Режим доступа: свободный.
2. Библиотека Системного Программиста. Том 8 Локальные сети персональных компьютеров. Использование протоколов IPX, SPX, NETBIOS [Электронный ресурс] / Электрон. дан. http://www.sources.ru/protocols/bsp08/ch2.htm – Режим доступа: свободный.
3. «Википедия» — свободная энциклопедия. Winsocket [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/Winsock – Режим доступа: свободный.
4. Библиотека Системного Программиста. Том 8. Локальные сети персональных компьютеров. Использование протоколов IPX, SPX, NETBIOS. [Электронный ресурс] / Электрон. дан. http://www.sources.ru/protocols/bsp08/index.html – Режим доступа: свободный.
5. Протоколы IPX/SPX [Электронный ресурс] / Электрон. дан. http://otherreferats.allbest.ru/radio/00199669_0.html – Режим доступа: свободный.
6. Протокол SPX [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/SPX – Режим доступа: свободный.
7. Microsoft MSDN [Электронный ресурс] / Электрон. дан. http://msdn.microsoft.com/ru-Ru/library/windows/desktop/ms741394(v=vs.85).aspx – Режим доступа: свободный.
8. Введение в ОС Linux [Электронный ресурс] / Электрон. дан. http://www.skif.bas-net.by/bsuir/base/base.html – Режим доступа: свободный.
9. «Википедия» — свободная энциклопедия. IP-протокол [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/IP-протокол – Режим доступа: свободный.

10. Клуб программистов [Электронный ресурс] / Электрон. дан. http://club.shelek.ru/viart.php?id=35 – Режим доступа: свободный.
11. Функции преобразования порядка байт [Электронный ресурс] / Электрон. дан. http://cs.mipt.ru/docs/courses/osstud/man/htons.htm – Режим доступа: свободный.
12. «Википедия» — свободная энциклопедия. TCP [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/TCP – Режим доступа: свободный.
13. «Википедия» — свободная энциклопедия. UDP [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/UDP – Режим доступа: свободный.
14. Протокол датаграмм клиента [Электронный ресурс] / Электрон. дан. http://citforum.ru/internet/tifamily/udpspec.shtml – Режим доступа: свободный.
15. Протокол UDP [Электронный ресурс] / Электрон. дан. http://book.itep.ru/4/44/udp_442 – Режим доступа: свободный.1
16. «Википедия» — свободная энциклопедия. ARP [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/ARP – Режим доступа: свободный.
17. ARP-протокол [Электронный ресурс] / Электрон. дан. http://веб-информ.рф/C++/5/060700.htm– Режим доступа: свободный.
18. Вычислительные системы, сети и коммуникации. Часть 7. ARP-протокол [Электронный ресурс] / Электрон. дан. http://iablov.narod.ru/net/net7gl.doc – Режим доступа: свободный.
19. Microsoft MSDN. IP Helper [Электронный ресурс] / Электрон. дан. http://msdn.microsoft.com/en-us/library/windows/desktop/aa366071(v=vs.85).aspx– Режим доступа: свободный.
20. «Википедия» — свободная энциклопедия. DNS [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/DNS – Режим доступа: свободный.
21. «Википедия» — свободная энциклопедия. DHCP [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/DHCP – Режим доступа: свободный.

22. «Википедия» — свободная энциклопедия. POP3 [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/POP3 – Режим доступа: свободный.
23. «Википедия» — свободная энциклопедия. SMTP [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/SMTP – Режим доступа: свободный.
24. Изучение протокола SMTP [Электронный ресурс] / Электрон. дан. http://plasma.karelia.ru/~alexmou/nets_tele/lab03.pdf – Режим доступа: свободный.
25. Протокол POP3[Электронный ресурс] / Электрон. дан. http://cdo.bseu.by/library/ibs1/applic_l/e_mail/internet/pop3.htm – Режим доступа: свободный.
26. «Википедия» — свободная энциклопедия. SNMP [Электронный ресурс] / Электрон. дан. http://ru.wikipedia.org/wiki/SNMP – Режим доступа: свободный.
27. Microsoft MSDN. SNMP Reference [Электронный ресурс] / Электрон. дан. http://msdn.microsoft.com/en-us/library/windows/desktop/aa379001(v=vs.85).aspx – Режим доступа: свободный.
28. SNMP: протокол управления сетью [Электронный ресурс] / Электрон. дан. http://www.laes.ru/list/pve/SNMP/tcp25.html#t257000 – Режим доступа: свободный.
29. Протокол управления SNMP [Электронный ресурс] / Электрон. дан. http://book.itep.ru/4/44/snm_4413.htm – Режим доступа: свободный.