Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра программного обеспечения информационных технологий

Сети ЭВМ

Лабораторный практикум для студентов специальности I-40 01 01
«Программное обеспечение информационных технологий»
дневной формы обучения

В 2-х частях

Часть1

Минск 2006

УДК 004.7 (075.8)
ББК 32.973. 20 я 73
С 33

Р е ц е н з е н т:
доцент кафедры ПО ЭС Высшего государственного колледжа связи,
канд. техн. наук А.А. Корбут

А в т о р ы:
Д.А. Сурков, С.В. Коростель, Е.В. Мельникова, И.М. Марина

С 33 **Сети ЭВМ:** Лабораторный практикум для студ. спец. I-40 01 01 «Программное обеспечение информационных технологий» дневн. формы обуч.: В 2 ч. Ч. 1 / Д.А. Сурков, С.В. Коростель, Е.В. Мельникова, И.М. Марина. – Мн.: БГУИР, 2006. – 36 с.

ISBN 985-444-964-5 (ч. 1)

В 1-й части издания представлена информация о межсетевом взаимодействии, рассмотрены методы и алгоритмы программирования сокетов

УДК 004.7 (075.8)
ББК 32.973. 20 я 73

ISBN 985-444-964-5 (ч. 1)
ISBN 985-444-963-7

Содержание

Лабораторная работа №1

Цель работы:

1. Изучить принципы работы сети Ethernet; кадр Ethernet; MAC-адрес.
2. Протокол IP. Адрес IP, сети и подсети, маска сети. Назначение протоколов ARP, RARP, ICMP. Маршрутизация (routing), шлюз (gateway), время жизни пакета TTL (на уровне понятий).
3. Ознакомится с протоколом NetBIOS. Назначение протокола NetBIOS и работа NetBIOS поверх TCP/IP.
4. Изучить сетевые команды оболочки Windows: arp, netstat, ipconfig, hostname, nslookup, ping, pathping, tracert, route, nbtstat. Принцип работы команд ping, tracert.

Краткие теоретические сведения

Семейство протоколов Transmission Control Protocol/Internet Protocol (TCP/IP) – это стандартный промышленный набор протоколов, разработанный для глобальных вычислительных сетей (Wide Area Networks, WAN).

Параметры конфигурации

Протокол TCP/IP использует IP-адрес, маску подсети и шлюз по умолчанию для соединения с узлами. Узлы TCP/IP, работающие в глобальной сети, требуют задания всех трех параметров в конфигурации. Каждая плата сетевого адаптера в компьютере, использующем TCP/IP, нуждается в этих параметрах.

IP-адрес

IP-адрес – это логический 32-разрядный адрес, однозначно определяющий узел TCP/IP. Каждый IP-адрес состоит из двух частей: идентификатора сети и идентификатора узла. Первый служит для обозначения всех узлов в одной физической сети. Второй обозначает конкретный узел сети. Каждому компьютеру, использующему TCP/IP, требуется уникальный IP-адрес.

Маска подсети

Маска подсети выделяет часть IP-адреса и позволяет TCP/IP отличить идентификатор сети от идентификатора узла. Пытаясь связаться, узлы TCP/IP используют маску подсети (например, 255.255.255.0), чтобы определить, находится узел-получатель в локальной или удаленной сети.

Шлюз по умолчанию

Для того чтобы установить соединение с узлом из другой сети, необходимо сконфигурировать IP-адрес шлюза по умолчанию. TCP/IP посылает пакеты, предназначенные для удаленных сетей, на шлюз по умолчанию, но только в том случае, если на локальном узле не сконфигурирован другой маршрут к сети получателя. Если не сконфигурирован шлюз по умолчанию, то связь может быть ограничена локальной сетью.

Тестирование TCP/IP при помощи утилит Ipconfig и Ping

После того как установлен протокол TCP/IP, полезно проверить и протестировать конфигурацию и все соединения с другими узлами TCP/IP и сетями. Это можно сделать утилитами Ipconfig и Ping.

Утилита Ipconfig

Можно использовать утилиту Ipconfig для проверки параметров конфигурации узла, включая IP-адрес, маску подсети и шлюз по умолчанию. Это полезно при выяснении, успешно ли прошла инициализация TCP/IP и не дублируется ли IP-адрес, указанный в конфигурации.

Синтаксис команды:

ipconfig

Если протокол инициализировался успешно с заданной конфигурацией, то на экране отобразятся IP-адрес, маска подсети и шлюз по умолчанию. Если адрес, заданный в конфигурации уже используется другим узлом в сети на том же сегменте, то отобразится заданный IP-адрес, но маска подсети будет равна 0.0.0.0

Утилита Ping

После проверки конфигурации утилитой Ipconfig можно запустить утилиту Ping (Packet InterNet Groper) для тестирования соединений. Это диагностическое средство тестирует конфигурации TCP/IP и позволяет определить неисправности соединения. Утилита Ping использует пакеты эхо-запроса (echo request) и эхо-ответа (echo reply) протокола ICMP (Internet Control Message Protocol) для проверки доступности и работоспособности определенного узла TCP/IP.

Синтаксис команды:

ping IP_адрес

Если проверка прошла успешно, то отобразится сообщение типа*:*

Pinging IP_адрес with 32 bytes of data:
Reply from IP_адрес: bytes= x time<10ms TTL= x
Reply from IP_адрес: bytes= x time<10ms TTL= x
Reply from IP_адрес: bytes= x time<10ms TTL= x
Reply from IP_адрес: bytes= x time<10ms TTL= x

Общие сведения об именах NetBIOS

Имя NetBIOS назначается компьютеру в сети Microsoft Windows. Стандарт NetBIOS, разработанный для IBM фирмой Sytek Corporation в 1983 году, позволяет приложениям взаимодействовать по сети. Этот стандарт определяет интерфейс сеансового уровня и протокол передачи данных и управления сеансом.

Интерфейс NetBIOS – доступный пользовательским приложениям стандартный прикладной интерфейс (API) для выполнения сетевого ввода/вывода и отправки команд управления к ПО нижележащего протокола. Прикладная программа, использующая интерфейс NetBIOS для сетевого взаимодействия, может работать с любым протоколом, поддерживающим интерфейс NetBIOS.

Стандартом NetBIOS определяется также протокол, действующий на сеансовом/транспортном уровне. Он реализуется в ПО нижележащего протокола, например NBFP (NetBEUI) или NetBT, где представлен весь набор команд сетевого ввода/вывода интерфейса NetBIOS. NetBIOS поверх TCP/IP, или просто NetBT, – это сетевая служба сеансового уровня.

NetBIOS поддерживает следующие команды и функции:

- регистрацию и проверку сетевых имен;
- запуск и завершение сеанса;
- надежную передачу данных сеанса, ориентированного на соединение;
- ненадежную передачу датаграмм (datagram) без установки соединения;
- возможность мониторинга и управления протоколом (драйвером) и адаптером.

Команда Arp

Команда arp используется для просмотра, добавления или удаления записей в таблицах трансляции адресов IP в физические адреса. Эти записи используются при работе протокола Address Resolution Protocol (ARP)

Синтаксис команды:

arp -a [inet_addr] [-N [if_addr]]
arp -d inet_addr [if_addr]
arp -s inet_addr ether_addr [if_addr]

Команда Netstat

Производит отображение активных подключений TCP портов, прослушиваемых компьютером, статистики протокола Ethernet, таблицы маршрутизации IP, статистики семейства протоколов IPv4 (для протоколов IP, ICMP, TCP и UDP) и семейства протоколов IPv6 (для протоколов IPv6, ICMPv6, TCP через IPv6 и UDP через IPv6). Запущенная без параметров команда netstat отображает подключения TCP.

Синтаксис команды:

netstat [-a] [-e] [-n] [-o] [-p протокол] [-r] [-s] [интервал]

Чтобы вывести все активные подключения, отсортированные по возрастанию номера порта, необходимо набрать:

netstat -n

Команда nslookup

Предоставляет сведения, предназначенные для диагностики инфраструктуры DNS. Для использования этого средства необходимо быть знакомым с принципами работы системы DNS. Средство командной строки nslookup доступно, только если установлен протокол TCP/IP

Синтаксис команды:

nslookup [-подкоманда ...] [{искомый_компьютер| [-сервер]}]

Команда Hostname

Утилита командной строки hostname выводит имя системы, на котором была запущена эта команда.

У данной команды нет параметров.

Команда Traceroute

Определяет путь до точки назначения с помощью посылки в точку назначения эхо-сообщений протокола Internet Control Message Protocol (ICMP) с постоянным увеличением значений срока жизни (Time to Live, TTL). Выведенный путь – это список ближайших адресов маршрутизаторов, находящихся на пути между узлом источника и точкой назначения. Ближний адрес представляют со-

бой адрес маршрутизатора, который является ближайшим к узлу отправителя на пути. Например, чтобы вывести трассу маршрута к http://www.microsoft.com, нужно выполнить команду:

tracert www.microsoft.com

Запущенная без параметров, команда **tracert** выводит справку.

Команда Route

Эта команда нужна для редактирования или просмотра таблицы маршрутов IP из командной строки. Параметр /? выводит все доступные параметры команды.

Задания к лабораторной работе №1

Программы пишутся на любом языке программирования, но должны использоваться **только** функции Windows API.

Написать программу, реализующую следующие функции:

1. Отображение MAC-адреса компьютера (можно воспользоваться функцией netbios).
2. Отображение всех рабочих групп, компьютеров в сети и их ресурсов (папок, открытых для общего доступа, принтеров). Воспользоваться функциями WNetXXX.

Пример программы получения MAC-адреса компьютера

```
typedef struct _ASTAT_
{
   ADAPTER_STATUS adapt;
   NAME_BUFFER    NameBuff [30];
}ASTAT, * PASTAT;

ASTAT Adapter;

// Функция получения MAC адреса.
// На вход получает указатель на буфер, куда записывается строковое
// представление полученного MAC адреса.
BOOL GetMacAddress(char *buffer)
{
   NCB ncb;
   UCHAR uRetCode;
   char NetName[50];

   memset( &ncb, 0, sizeof(ncb) );
   ncb.ncb_command = NCBRESET;
   ncb.ncb_lana_num = 0;

   uRetCode = Netbios( &ncb );
```

```
    memset( &ncb, 0, sizeof(ncb) );
    ncb.ncb_command = NCBASTAT;
    ncb.ncb_lana_num = 0;

    strcpy( (char *) ncb.ncb_callname,  "*               " );
    ncb.ncb_buffer = (unsigned char *) &Adapter;
    ncb.ncb_length = sizeof(Adapter);

    uRetCode = Netbios( &ncb );
    if ( uRetCode == 0 )
    {
       sprintf(buffer, "%02X-%02X-%02X-%02X-%02X-%02X\n",
             Adapter.adapt.adapter_address[0],
             Adapter.adapt.adapter_address[1],
             Adapter.adapt.adapter_address[2],
             Adapter.adapt.adapter_address[3],
             Adapter.adapt.adapter_address[4],
             Adapter.adapt.adapter_address[5] );
       return TRUE;
    }
    return FALSE;
}
```

Пример программы перечисления компьютеров в локальной сети

```
BOOL WINAPI EnumerateFunc(HWND hwnd, HDC hdc, LPNETRESOURCE lpnr)
{
  DWORD dwResult, dwResultEnum;
  HANDLE hEnum;
  DWORD cbBuffer = 16384;
  DWORD cEntries = -1;        // Искать все объекты
  LPNETRESOURCE lpnrLocal;
  DWORD i;

  // Вызов функции WNetOpenEnum для начала перечисления компьютеров.
  dwResult = WNetOpenEnum(RESOURCE_GLOBALNET, // все сетевые ресурсы
                    RESOURCETYPE_ANY,   // все типы ресурсов
                    0,       // перечислить все ресурсы
                    lpnrLocal,     // равно NULL при первом вызове функции
                    &hEnum);  // дескриптор ресурса

  if (dwResult != NO_ERROR)
  {
    // Обработка ошибок.
    NetErrorHandler(hwnd, dwResult, (LPSTR)"WNetOpenEnum");
```

```
  return FALSE;
}

// Вызвов функции GlobalAlloc для выделения ресурсов.
lpnrLocal = (LPNETRESOURCE) GlobalAlloc(GPTR, cbBuffer);
if (lpnrLocal == NULL)
   return FALSE;

do
{

 // Инициализируем буфер.
 ZeroMemory(lpnrLocal, cbBuffer);

 // Вызов функции WNetEnumResource для продолжения перечисления
 // доступных ресурсов сети.
 dwResultEnum = WNetEnumResource(hEnum,
                        &cEntries,  // Определено выше как -1
                        lpnrLocal,
                        &cbBuffer); // размер буфера

 // Если вызов был успешен, то структуры обрабатываются циклом.
 if (dwResultEnum == NO_ERROR)
 {
  for(i = 0; i < cEntries; i++)
  {
   // Вызов определенной в приложении функции для отображения
   // содержимого структур NETRESOURCE.
   DisplayStruct(hdc, &lpnrLocal[i]);

   //  Если структура NETRESOURCE является контейнером, то
   //  функци\ EnumerateFunc вызывается рекурсивно.
   if(RESOURCEUSAGE_CONTAINER == (lpnrLocal[i].dwUsage
                          & RESOURCEUSAGE_CONTAINER))
     if(!EnumerateFunc(hwnd, hdc, &lpnrLocal[i]))
       TextOut(hdc, 10, 10, "EnumerateFunc returned FALSE.", 29);
  }
 }

 // Обработка ошибок.
 else if (dwResultEnum != ERROR_NO_MORE_ITEMS)
 {
  NetErrorHandler(hwnd, dwResultEnum, (LPSTR)"WNetEnumResource");
  break;
 }
```

```
  }
  while(dwResultEnum != ERROR_NO_MORE_ITEMS);

  // Вызов функции GlobalFree для очистки ресурсов.
  GlobalFree((HGLOBAL)lpnrLocal);

  // Вызов WNetCloseEnum для остановки перечисления.
  dwResult = WNetCloseEnum(hEnum);

  if(dwResult != NO_ERROR)
  {
    // Обработка ошибок.
    NetErrorHandler(hwnd, dwResult, (LPSTR)"WNetCloseEnum");
    return FALSE;
  }
  return TRUE;
}
```

Лабораторная работа № 2

Цель работы: изучить архитектуру прикладного интерфейса Windows Sockets (Winsock).

Краткие теоретические сведения

Протокол TCP

Протокол TCP предоставляет надежную, ориентированную на соединение службу доставки.

Данные протокола TCP передаются сегментами, и соединение должно быть установлено до того, как узлы начнут обмениваться данными. TCP использует потоки, в которых данные представлены в виде последовательности байт.

TCP обеспечивает надежность, присваивая номера последовательности (sequence number) каждому передаваемому сегменту. Если сегмент разбивается на мелкие пакеты, то узел-получатель сможет узнать, все ли части получены. Для этого используются подтверждения. Для каждого отправленного сегмента узел-получатель должен вернуть отправителю подтверждение (acknowledgement, ACK) в течение определенного времени.

Если отправитель не получил ACK, то данные передаются повторно. Если сегмент поврежден, то узел-получатель отвергает его. Поскольку ACK в этом случае не посылается, отправитель передает сегмент еще раз.

Примечание: TCP описан в стандарте RFC 793

Порты

Приложения, использующие «сокеты», идентифицируют себя на компьютере посредством номера порта (port number). Например, FTP-сервер использует определенный TCP-порт, поэтому другие приложения могут связаться с ним.

Порты могут иметь любой номер от 0 до 65 536. Номера портов для приложений клиентов динамически назначаются операционной системой при обработке запроса на обслуживание. Существуют зарезервированные (или заранее известные, от англ. well-known) номера портов, которые закреплены за стандартными прикладными протоколами и службами. Их список можно узнать, просмотрев текстовый файл Windows\System32\Drivers\Etc\Services.

Номера зарезервированных портов расположены в интервале от 1 до 1024. Полный список зарезервированных номеров портов определен в стандарте RFC 1700.

«Сокеты»

«Сокет» (от англ. socket) во многом аналогичен дескриптору файла (file handle). Он обеспечивает конечную точку сетевого соединения. Приложение, создавая «сокет», указывает три параметра: IP-адрес узла, тип обслуживания (протокол TCP для ориентированного на соединение обслуживания и UDP для не ориентированного) и порт, используемый приложением.

Протокол UDP

Протокол User Datagram Protocol (UDP) обеспечивает не ориентированную на соединение службу доставки датаграмм по принципу «максимального усилия». Это означает, что получение всей датаграммы или правильной последовательности отправленных пакетов не гарантируется.

Протокол UDP используется приложениями, не требующими подтверждения. Обычно такие приложения передают данные небольшого объема за один раз. Примеры служб и приложений, использующих UDP: сервис имен NetBIOS, сервис датаграмм NetBIOS и сервис SNMP.

Порты протокола UDP

Для использования протокола UDP приложение должно знать IP-адрес и номер порта получателя. Порт – место назначения при доставке сообщений – идентифицируется уникальным номером. Порт действует как мультиплексная очередь сообщений, то есть он может получать несколько сообщений одновременно. Важно отметить, что порты протокола UDP, перечисленные в таблице, отличаются от портов TCP несмотря на использование тех же значений номеров. Протокол UDP описан в стандарте RFC 768.

Сокеты в ОС Windows

В ОС Windows для работы с «сокетами» используется библиотека Winsock, которая представляют собой высокоуровневый унифицирован-ный интерфейс взаимодействия с телекоммуникационными протоколами. Основное подспорье в изучении сокетов – Windows Sockets 2 SDK (от англ. Software Development Kit) – документация, набор заголовочных файлов и инструментарий разработчика.

Библиотека Winsock поддерживает два вида сокетов – синхронные (блокируемые) и асинхронные (неблокируемые). Синхронные сокеты задерживают

управление на время выполнения операции, а асинхронные возвращают его немедленно, продолжая выполнение в фоновом режиме, и, закончив работу, уведомляют об этом вызывающий код.

Независимо от вида, сокеты делятся на два типа – потоковые и дейтаграммные. Потоковые сокеты работают с установкой соединения, обеспечивая надежную идентификацию обоих сторон и гарантируют целостность и успешность доставки данных. Дейтаграмные сокеты работают без установки соединения и не обеспечивают ни идентификации отправителя, ни контроля успешности доставки данных, зато они заметно быстрее потоковых.

Выбор того или иного типа сокетов определяется транспортным протоколом на котором работает сервер, – клиент не может по своему желанию установить с дейтаграммным сервером потоковое соединение.

Замечание: дейтаграммные сокеты опираются на протокол UDP, а потоковые на TCP.

Перед началом использования функций библиотеки Winsock ее необходимо подготовить к работе вызовом функции

int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData)

передав в старшем байта слова wVersionRequested номер требуемой версии, а в младшем – номер подверсии.

Второй шаг – создание объекта "сокет". Это осуществляется функцией

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

Первый слева аргумент указывает на семейство используемых протоколов. Для интернет-приложений он должен иметь значение AF_INET. Следующий аргумент задает тип создаваемого сокета – потоковый (SOCK_STREAM) или дейтаграммный (SOCK_DGRAM). Последний аргумент уточняет, какой транспортный протокол следует использовать. Нулевое значение соответствует выбору по умолчанию: TCP для потоковых сокетов и UDP для дейтаграммных. Если функция завершилась успешно, она возвращает дескриптор сокета, в противном случае значение INVALID_SOCKET.

Дальнейшие шаги зависит от того, является приложение сервером или клиентом. Ниже эти два случая будут описаны раздельно.

Клиент: шаг третий – для установки соединения с удаленным узлом потоковый сокет должен вызвать функцию

int connect (SOCKET s, const struct sockaddr FAR* name, int namelen)

Датаграмные сокеты работают без установки соединения, поэтому, обычно не обращаются к функции connect.

Первый слева аргумент – дескриптор сокета, возращенный функцией socket; второй – указатель на структуру "sockaddr", содержащую в себе адрес и порт удаленного узла с которым устанавливается соединение. Последний аргумент сообщает функции размер структуры sockaddr.

После вызова connect система предпринимает попытку установить соединение с указанным узлом. Если по каким-то причинам это сделать не удастся (адрес задан неправильно, узел не существует или "висит", компьютер находится не в сети), функция возвратит ненулевое значение.

Сервер: шаг третий – прежде, чем сервер сможет использовать сокет, он должен связать его с локальным адресом. Локальный, как, впрочем, и любой другой адрес Интернета, состоит из IP-адреса узла и номера порта. Если сервер имеет несколько IP адресов, то сокет может быть связан как со всеми сразу (для этого вместо IP-адреса следует указать константу INADDR_ANY равную нулю), так и с каким-то конкретным одним.

Связывание осуществляется вызовом функции

int bind (SOCKET s, const struct sockaddr FAR* name, int namelen)

Первым слева аргументом передается дескриптор сокета, возращенный функций socket, за ним следуют указатель на структуру sockaddr и ее длина.

Строго говоря, клиент также должен связывать сокет с локальным адресом перед его использованием, однако, за него это делает функция connect, ассоциируя сокет с одним из портов, наугад выбранных из диапазона 1024-5000. Сервер же должен "садиться" на заранее определенный порт, например, 21 для FTP, 23 для telnet, 25 для SMTP, 80 для WEB, 110 для POP3 и т.д. Поэтому ему приходится осуществлять связывание "вручную".

При успешном выполнении функция возвращает нулевое значение и ненулевое в противном случае.

Сервер: шаг четвертый – выполнив связывание, потоковый сервер переходит в режим ожидания подключений, вызывая функцию

int listen (SOCKET s, int backlog),

где s – дескриптор сокета, а backlog – максимально допустимый размер очереди сообщений.

Размер очереди ограничивает количество одновременно обрабатываемых соединений, поэтому, к его выбору следует подходить "с умом". Если очередь полностью заполнена, очередной клиент при попытке установить соединение получит отказ (TCP пакет с установленным флагом RST). В то же время максимально разумное количество подключений определяются производительностью сервера, объемом оперативной памяти и т.д.

Датаграммные серверы не вызывают функцию listen, т.к. работают без установки соединения и сразу же после выполнения связывания могут вызывать recvfrom для чтения входящих сообщений, минуя четвертый и пятый шаги.

Сервер: шаг пятый – извлечение запросов на соединение из очереди осуществляется функцией

SOCKET accept (SOCKET s, struct sockaddr FAR* addr, int FAR* addrlen),

которая автоматически создает новый сокет, выполняет связывание и возвращает его дескриптор, а в структуру sockaddr заносит сведения о подключившемся клиенте (IP-адрес и порт). Если в момент вызова accept очередь пуста, функция не возвращает управление до тех пор, пока с сервером не будет установлено хотя бы одно соединение. В случае возникновения ошибки функция возвращает отрицательное значение.

Для параллельной работы с несколькими клиентами следует сразу же после извлечения запроса из очереди порождать новый поток (процесс), передавая ему дескриптор созданного функцией accept сокета, затем вновь извлекать из

очереди очередной запрос и т.д. В противном случае, пока не завершит работу один клиент, север не сможет обслуживать всех остальных.

Клиент и сервер: после того как соединение установлено, потоковые сокеты могут обмениваться с удаленным узлом данными, вызывая функции

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

int recv (SOCKET s, char FAR* buf, int len, int flags)

для посылки и приема данных соответственно.

Функция send возвращает управление сразу же после ее выполнения независимо от того, получила ли принимающая сторона наши данные или нет. При успешном завершении функция возвращает количество передаваемых (не переданных!) данных – т.е. успешное завершение еще не свидетельствует об успешной доставке! В общем-то, протокол TCP (на который опираются потоковые сокеты) гарантирует успешную доставку данных получателю, но лишь при условии, что соединение не будет преждевременно разорвано. Если связь прервется до окончания пересылки, данные останутся не переданными, но вызывающий код не получит об этом никакого уведомления! А ошибка возвращается лишь в том случае, если соединение разорвано до вызова функции send.

Функция же recv возвращает управление только после того, как получит хотя бы один байт. Точнее говоря, она ожидает прихода целой дейтаграммы. Дейтаграмма – это совокупность одного или нескольких IP пакетов, посланных вызовом send. Упрощенно говоря, каждый вызов recv за один раз получает столько байтов, сколько их было послано функцией send. При этом подразумевается, что функции recv предоставлен буфер достаточных размеров, – в противном случае ее придется вызвать несколько раз. Однако, при всех последующих обращениях данные будут браться из локального буфера, а не приниматься из сети, т.к. TCP-провайдер не может получить "кусочек" дейтаграммы, а только ею всю целиком.

Дейтаграммный сокет так же может пользоваться функциями send и recv, если предварительно вызовет connect (см. "Клиент: шаг третий"), но у него есть и свои, "персональные", функции:

int sendto (SOCKET s, const char FAR * buf, int len,int flags, const struct sockaddr FAR * to, int tolen)

int recvfrom (SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen)

Они очень похожи на send и recv, – разница лишь в том, что sendto и recvfrom требуют явного указания адреса узла принимаемого или передаваемого данные. Вызов recvfrom не требует предварительного задания адреса передающего узла – функция принимает все пакеты, приходящие на заданный UDP-порт со всех IP адресов и портов. Напротив, отвечать отправителю следует на тот же самый порт откуда пришло сообщение. Поскольку, функция recvfrom заносит IP-адрес и номер порта клиента после получения от него сообщения, программисту фактически ничего не нужно делать – только передать sendto тот же самый указатель на структуру sockaddr, который был ранее передан функции recvfrem, получившей сообщение от клиента.

Еще одна деталь – транспортный протокол UDP, на который опираются дейтаграммные сокеты, не гарантирует успешной доставки сообщений и эта задача ложиться на плечи самого разработчика. Решить ее можно, например, посылкой клиентом подтверждения об успешности получения данных. Правда, клиент тоже не может быть уверен, что подтверждение дойдет до сервера, а не потеряется где-нибудь в дороге. Подтверждать же получение подтверждения - бессмысленно, т. к. это рекурсивно неразрешимо. Лучше вообще не использовать дейтаграммные сокеты на ненадежных каналах.

Во всем остальном обе пары функций полностью идентичны.

Все четыре функции при возникновении ошибки возвращают значение SOCKET_ERROR = -1.

Шаг шестой, последний – для закрытия соединения и уничтожения сокета предназначена функция "int closesocket (SOCKET s)", которая в случае удачного завершения операции возвращает нулевое значение.

Перед выходом из программы, необходимо вызвать функцию "int WSACleanup (void)" для деинициализации библиотеки WINSOCK и освобождения используемых этим приложением ресурсов. Внимание: завершение процесса функцией ExitProcess автоматически не освобождает ресурсы сокетов.

Задания к лабораторной работе № 2

Все программы выполняются на любом языке программирования, используются **только** функции Windows API. Для каждого варианта должно быть реализовано две версии программы: одна работающая посредством протокола TCP, вторая – UDP.

Написать программу, реализующую следующие функции:

Вариант 1: Программа представляет простейший «чат» (от англ. chat), позволяющий обмениваться текстовыми сообщениями между двумя компьютерами.

Вариант 2: Программа позволяет копировать бинарные файлы с одного компьютера на другой.

Вариант 3: Программа измеряет скорость передачи информации по протоколам TCP и UDP, а так же количество потерянных (искаженных) пакетов. Трафик генерируется псевдослучайным образом (т.е. генерируется псевдослучайная последовательность данных, отсылается на другой компьютер и там сравнивается с эталоном).

Вариант 4: Программа удаленного рисования. На первом компьютере пользователь может рисовать кривые мышкой на «холсте». На втором на таком же холсте рисунок повторяется в реальном времени.

Пример программы реализации сервера DayTime

```
#include <stdio.h>
#include <sys/types.h>
```

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define fatal(x) { perror(x); exit(1); }

main()
{
        int    s, c, sz;
        struct sockaddr_in ssa, csa;
        struct sockaddr *sp, *cp;
        char   *tstr, node[128], service[16];
        time_t itime;

        sp = (struct sockaddr *)&ssa;
        cp = (struct sockaddr *)&csa;

        /* Создаём сокет */
        s = socket(AF_INET, SOCK_STREAM, 0);
        if (s == -1)
                fatal("Невозможно создать сокет");

        /* Занимаем порт 13 */
        ssa.sin_family = AF_INET;
        ssa.sin_port = htons(13);
        ssa.sin_addr.s_addr = INADDR_ANY;

        if (bind(s, sp, sizeof(ssa)) == -1)
                fatal("Невозможно занять порт");

        /*
         * Переводим сокет в режим ожидания запросов
         * на установление соединения
         */
        if (listen(s, 0) == -1)
                fatal("Ошибка при выполнении listen");

        while (1) {
                /* Принимаем соединение */
                sz = sizeof(csa);
                if ((c = accept(s, cp, &sz)) == -1)
                        fatal("Ошибка при выполнении accept");

                /* Получаем строку, содержащую дату и время */
                itime = time(NULL);
```

```
            tstr = ctime(&itime);

            /* Выводим время поступления запроса */
            printf("%s\tполучен запрос от ", tstr);
            /* Выводим информацию о клиенте */
            if (getnameinfo(cp, sz, node,
                        128, service, 16, 0) == 0)
                printf("%s:%s\n", node, service);
            else
                printf("%s:%d\n", inet_ntoa(csa.sin_addr),
                    ntohs(csa.sin_port));

            /* Отправляем дату и время клиенту */
            send(c, tstr, 25, 0);

            /* Закрываем соединение */
            close(c);
        }
}
```

Лабораторная работа № 3

Цель работы: изучить протоколы Echo, Time, DayTime, WhoIs, Finger, RLogin, Telnet.

Краткие теоретические сведения

Протокол Echo

Эхо-сервис весьма полезен для отладки и выполнения измерений. Этот сервис просто возвращает отправителю любые данные, полученные от него. Полное описание протокола можно найти в стандарте RFC 862.

Службы Echo на базе протокола TCP.

Один из вариантов эхо-сервиса определен как основанный на организации соединений приложение TCP. Эхо-сервер прослушивает соединения TCP на порту 7. После организации соединения все полученные через это соединение данные возвращаются отправителю. Процесс возврата полученных данных отправителю продолжается до тех пор, пока инициатор соединения не разорвет это соединение.

Службы Echo на базе протокола UDP.

Другой вариант эхо-сервиса не использует прямых соединений и основан на передаче дейтаграмм UDP. Эхо-сервер прослушивает порт 7 (UDP) и возвращает отправителю все принятые через этот порт дейтаграммы.

Time

Данный протокол предназначен для синхронизации времени. В сети работают time-серверы, у которых можно запросить точное время. Следует заме-

тить, что в настоящее время для синхронизации времени в глобальных сетях используется более сложный протокол - NTP - Network Time Protocol. В ответ на запрос клиента, сервер возвращает время в секундах (32х битное двоичное число), прошедшее с 00:00:00 1 января 1900 года.

Этот протокол может использовать в качестве транспортной службы как UDP-протокол, так и TCP-протокол. Стандартный порт протокола - 37.

Если в качестве транспортной службы используется TCP, взаимодействие осуществляется так:

SERVER: прослушивает 37 порт, ожидая соединений
CLIENT: запрашивает соединение с портом 37 сервера
SERVER: посылает время в виде двоичного 32х битного числа
CLIENT: получает время
SERVER: закрывает соединение
CLIENT: закрывает соединение

Если сервер по каким-либо причинам не может определить время на своей стороне, он отказывается от соединения, не посылая ничего.

Если в качестве транспортной службы используется UDP, взаимодействие осуществляется так:

SERVER: прослушивает 37 порт, ожидая соединений
CLIENT: посылает серверу пустой UDP-пакет, номер порта = 37
SERVER: получает пустой UDP-пакет
SERVER: посылает UDP-пакет, содержащий время в виде двоичного 32х битного числа
CLIENT: получает UDP-пакет, содержащий время

Если сервер по каким-либо причинам не может определить время на своей стороне, он отбрасывает полученный пустой UDP-пакет и не посылает ничего в ответ.

DayTime

Протокол DayTime определен в документе RFC867. Протокол может использовать в качестве транспортного протокола UDP и TCP. В случае использования UDP сервер занимает 13-й порт и ожидает поступления датаграмм. После получения датаграммы он отправляет по обратному адресу пакет, содержащий текущие дату и время в виде текстовой строки. Дополнительно, сервер выводит информацию о поступившем запросе на стандартный вывод. В случае использования TCP, как и в UDP, сервер занимает порт 13 и ожидает поступления запросов на установление соединения. После установления соединения, сервер отправляет клиенту строку, содержащую дату и время, и закрывает соединение.

WhoIs

Протокол Whois это информационный сервис. Несмотря на то, что любой узел может предоставить Whois сервис, наиболее широко используется InterNIC, rs.internic.net. Этот сервер содержит информацию обо всех зарегистрированных DNS доменах и о большинстве системных администраторов, которые ответственны за системы, подключенные к Internet. (Еще один подобный сервер nic.ddn.mil содержит информацию о сети MILNET.) К сожалению, не всегда

предоставляется полная информация. RFC 954 [Harrenstein, Stahl, and Feinler 1985] документирует сервис Whois.

С точки зрения протокола, сервер Whois работает с заранее известным портом TCP 43. Он принимает от клиента запрос на соединение, после чего клиент отправляет на сервер запрос длиной в 1 строку. Сервер выдает информацию и закрывает соединение. Запросы и отклики передаются в формате NVT ASCII. Он практически идентичен серверу Finger, за исключением того, что запросы и отклики содержат разную информацию.

Широко используемый Unix клиент – программа whois, однако можно использовать Telnet и ввести команды самостоятельно. Сначала отправляется запрос, содержащий знак вопроса, на что возвращается более подробная информация о поддерживаемых запросах клиента.

Finger

Протокол Finger возвращает информацию об одном или нескольких пользователях на указанном хосте. Это приложение обычно используется, для того чтобы посмотреть, находится ли конкретный пользователь в настоящее время в системе, или чтобы получить имя какого-либо пользователя, чтобы послать ему почту. RFC 1288 [Zimmerman 1991] описывает этот протокол.

Многие узлы не запускают Finger сервер по двум причинам. Во-первых, ошибки в программировании в ранних версиях сервера были одной из точек входа "червяка" в Internet в 1988 году. Во-вторых, протокол Finger может предоставить подробную информацию о пользователях (имя входа в систему, телефонные номера, время последнего входа и так далее), а эту информацию большинство администраторов считают частной. Раздел 3 RFC 1288 детально описывает аспекты секретности, соответствующие сервису Finger.

Сервер Finger использует заранее известный порт 79. Клиент осуществляет активное открытие на этот порт и отправляет запрос длиной в 1 строку. Сервер обрабатывает запрос, посылает назад вывод и закрывает соединение. Запрос и отклик в формате NVT ASCII, почти так же как мы видели в случае FTP и SMTP.

Обычно большинство пользователей Unix получают доступ к серверу Finger с использованием клиента finger, однако можно воспользоваться Telnet клиентом. Если запрос клиента состоит из пустой строки (которая в NVT ASCII передается как CR, за которой следует LF), это воспринимается как запрос на информацию о всех текущих пользователях.

RLogin

Rlogin появился в 4.2BSD и был предназначен для захода удаленным терминалом между Unix хостами. Поэтому Rlogin проще, чем Telnet, так как он не требует определения параметров, которые для одной и той же операционной системы известны заранее и для клиента, и для сервера. Через несколько лет Rlogin был перенесен на не-Unix системы.

RFC 1282 [Kantor 1991] содержит спецификацию протокола Rlogin. Однако, как и в случае с RFC посвященным RIP (Routing Information Protocol), он был написан после того, как Rlogin уже использовался в течении нескольких лет.

Rlogin использует одно TCP соединение между клиентом и сервером. После того как TCP соединение установлено, между клиентом и сервером осуществляется следующая последовательность действий.

Клиент отправляет серверу четыре строки: нулевой байт, имя пользователя на хосте клиента, заканчивающееся нулевым байтом, имя пользователя на хосте сервера, заканчивающееся нулевым байтом, тип терминала пользователя, за которым следует слэш, затем следует скорость терминала, и все это заканчивается нулевым байтом. Необходимо отправить именно два имени пользователя, потому что пользователи не обязаны иметь одинаковые имена на разных хостах.

Тип терминала передается от клиента к серверу, потому что эта информация необходима для большинства полноэкранных приложений. Скорость терминала передается, потому что некоторые приложения работают по-разному в зависимости от скорости.

Сервер отвечает нулевым байтом.

У сервера есть опция, с помощью которой он просит ввести пароль. Это осуществляется как обычный обмен данными по Rlogin соединению – специальные протоколы не применяются. Сервер отправляет клиенту строку (которую клиент отображает на терминале), чаще всего эта строка выглядит как «Password:». Если клиент не вводит пароль в течение определенного времени (обычно 60 секунд), сервер закрывает соединение. Все что вводится в ответ на приглашение сервера ввести пароль, передается в виде открытого текста. Символы введенного пароля посылаются так, как они есть. Каждый, кто может прочитать пакеты в сети, может прочитать любой пароль.

Сервер обычно отправляет запрос клиенту, спрашивая размер окна терминала.

Клиент посылает за один раз серверу 1 байт, каждый байт сервер отражает эхо-откликом. Функционально все довольно просто: то, что вводит пользователь, отправляется на сервер, а то, что сервер отправляет клиенту, отображается на терминале.

Telnet

Telnet был разработан, для того чтобы работать между хостами работающими под управлениием любых операционных систем, а также с любыми терминалами. Его спецификация, приведенная в RFC 854 [Postel and Reynolds 1983a], определяет терминал, который может являться наиболее общим, и который называется виртуальным сетевым терминалом (NVT - network virtual terminal). NVT это воображаемое устройство, находящееся на обоих концах соединения, у клиента и сервера, с помощью которого устанавливается соответствие между их реальными терминалами. Таким образом, операционная система клиента должна определять соответствие между тем типом терминала, за которым работает пользователь, с NVT. В свою очередь, сервер должен устанавливать соответствие между NVT и теми типами терминалов, которые он (сервер) поддерживает.

NVT это символьное устройство с клавиатурой и принтером. Данные, введенные пользователем с клавиатуры, отправляются серверу, а данные, полу-

ченные от сервера, поступают на принтер. По умолчанию клиент отражает эхом на принтер все, что ввел пользователь, однако, ниже мы увидим что, существуют опции, которые позволяют изменить подобное поведение.

Задания к лабораторной работе № 3

Все программы выполняются на любом языке программирования, возможно использование как функций Windows API, так и любых других библиотек работы с сокетами.

Написать программу, реализующую следующие функции клиента и сервера одного из протоколов:

Вариант 1: Протокол Echo.
Вариант 2: Протокол Time.
Вариант 3: Протокол DayTime.
Вариант 4: Протокол WhoIs.
Вариант 5: Протокол Finger.
Вариант 6: Протокол RLogin.
Вариант 7: Протокол Telnet.

Программа должна производить полную обработку команд запросов и ответов каждого из протоколов. Ввиду сложности вариантов 6 и 7 допускается обработка только основного подмножества команд.

Сдача всех программ должна сопровождаться демонстрацией работы не только написанного самостоятельно клиента с сервером, но и демонстрации возможности взаимодействия написанного клиента (сервера) с сервером (клиентом) стороннего производителя (это необходимо для того, что бы проверить что протокол, реализованный самостоятельно, совместим со стандартом).

Лабораторная работа №4

Цель работы: ознакомиться с семейством протоколов прикладного уровня:

1. and D. Reed, May 1993
2. File Transfer Protocol (FTP), RFC 959, by J. Postel and J. Reynolds, October 1985.
3. Hypertext Transfer Protocol version 1.0 (HTTP/1.0), RFC 1945.
4. Internet Relay Chat Protocol (IRC), RFC 1459, by J. Oikarinen Hypertext Transfer Protocol version 1.1 (HTTP/1.1), RFC 2616.
5. The Secure HyperText Transfer Protocol, RFC 2660.
6. Post Office Protocol Version 3, RFC 1939, by J. Myers, May 1996.
7. Simple Mail Transfer Protocol (SMTP), RFC 821.
8. SMTP Service Extensions, RFC 1869.
9. SMTP Service Extension for Authentication, RFC 2554.
10. Network News Transfer Protocol, RFC 977, by Brian Kantor and Phil Lapsley, February 1986.
11. Common NNTP Extensions, RFC 2980, by S. Barber, October 2000.

Краткие теоретические сведения

Протокол HTTP

Hypertext Transfer Protocol (HTTP, протокол пересылки гипертекста) – это язык, которым клиенты и серверы World Wide Web пользуются для общения между собой. Он, по сути дела, является основой в Web. Хотя HTTP в большей степени относится к сфере программирования серверов и клиентов, знание этого протокола важно и для CGI-программирования. Кроме того, иногда HTTP фильтрует информацию и передает ее обратно пользователям – это происходит, например, когда в окне браузера отображаются коды ошибок сервера.

Все HTTP-транзакции имеют один общий формат. Каждый запрос клиента и ответ сервера состоит из трех частей: строки запроса (ответа), раздела заголовка и тела. Клиент инициирует транзакцию следующим образом:

1. Клиент устанавливает связь с сервером по назначенному номеру порта (по умолчанию – 80). Затем клиент посылает запрос документа, указав HTTP-команду, называемую методом, адрес документа и номер версии HTTP. Например, в запросе

GET /index.html HTTP/1.0

используется метод GET, которым с помощью версии 1.0 HTTP запрашивается документ index.html. Методы HTTP более подробно рассматриваются ниже.

2. Клиент посылает информацию заголовка (необязательную), чтобы сообщить серверу информацию о своей конфигурации и данные о форматах документов, которые он может принимать. Вся информация заголовка указывается построчно, при этом в каждой строке приводится имя и значение. Например, приведенный ниже заголовок, посланный клиентом, содержит его имя и номер версии, а также информацию о некоторых предпочтительных для клиента типах документов:

User-Agent: Mozilla/4.05 (WinNT; 1)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

Завершается заголовок пустой строкой.

3. Послав запрос и заголовки, клиент может отправить и дополнительные данные. Эти данные используются главным образом теми CGI-программами, которые применяют метод POST. Клиенты также могут использовать их для помещения отредактированной страницы обратно на Web-сервер.

Сервер отвечает на запрос клиента следующим образом:

1. Первая часть ответа сервера – строка состояния, содержащая три поля: версию HTTP, код состояния и описание. Поле версии содержит номер версии HTTP, которой данный сервер пользуется для передачи ответа.

Код состояния – это трехразрядное число, обозначающее результат обработки сервером запроса клиента. Описание, следующее за кодом состояния, представляет собой просто понятный для человека текст, поясняющий код состояния. Например, строка состояния

HTTP/1.0 200 OK

говорит о том, что сервер для ответа использует версию HTTP 1.0. Код состояния 200 означает, что запрос клиента был успешным и затребованные данные будут переданы после заголовков.

2. После строки состояния сервер передает клиенту информацию заголовка, содержащую данные о самом сервере и затребованном документе. Ниже приведен пример заголовка:

Date: Fri, 10 Jan 2004 07:34:28 GMT
Server: Apache/2.2.6
Last-modified: Mon, 12 Jun 2003 22:54:48 GMT
Content-type: text/html
Content-length: 2482

Завершают заголовок 2 пустые строки.

3. Если запрос клиента успешен, то посылаются затребованные данные. Это может быть копия файла или результат выполнения CGI-программы. Если запрос клиента удовлетворить нельзя, передаются дополнительные данные в виде понятного для пользователя разъяснения причин, по которым сервер не смог выполнить данный запрос.

В HTTP 1.0 за передачей сервером затребованных данных следует разъединение с клиентом, и транзакция считается завершенной, если не передан заголовок Connection: Keep Alive.

Запросы клиента разбиваются на три раздела. Первая строка сообщения всегда содержит HTTP-команду, называемую методом, URI, который обозначает запрашиваемый клиентом файл или ресурс, и номер версии HTTP. Следующие строки запроса клиента содержат информацию заголовка. Информация заголовка содержит сведения о клиенте и информационном объекте, который он посылает серверу. Третья часть клиентского запроса представляет собой тело содержимого – собственно данные, посылаемые серверу.

Метод – это HTTP-команда, с которой начинается первая строка запроса клиента. Метод сообщает серверу о цели запроса. Для HTTP определены три основных метода: GET, HEAD и POST. Определены и другие методы, но они не так широко поддерживаются серверами, как три перечисленных. При задании имен методов учитывается регистр, поэтому «GET» и «get» различаются.

Метод GET

GET – это запрос информации, расположенной на сервере по указанному URL. GET – наиболее распространенный метод поиска с помощью браузеров документов для визуализации. Результат запроса GET может представлять собой, например, файл, доступный для сервера, результат выполнения программы или CGI-сценария, выходную информацию аппаратного устройства и т.д.

Если клиент пользуется в своем запросе методом GET, сервер отвечает строкой состояния, заголовками и затребованными данными. Если сервер не

может обработать запрос вследствие ошибки или отсутствия полномочий, он, как правило, посылает в информационном разделе ответа текстовое пояснение.

Тело информационного содержимого запроса GET всегда пустое. GET в переводе на человеческий язык означает примерно следующее: "Дайте мне этот файл". Для идентификации указанных в запросе клиента файла или программы обычно используется полное имя этого объекта на сервере.

Ниже приведен пример успешного запроса GET на получение файла. Клиент посылает запрос:

GET /index.html HTTP/1.O
Connection: Keep-Alive
User-Agent: Mozilla/4.05 (WinNT; 1)
Host: www.ora.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

Сервер отвечает:

HTTP/1.0 200 Document follows
Date: Fri, 20 Jan 2004 08:17:58 GMT
Server: Apache/1.2.6
Last-modified: Mon, 20 Jun 2003 21:53:08 GMT
Content-type: text/html
Content-length: 2482
(далее следует тело документа)

Метод POST

Метод POST позволяет посылать на сервер данные в запросе клиента. Эти данные направляются в программу обработки данных, к которой сервер имеет доступ (например, в CGI-сценарий). Метод POST может использоваться во многих приложениях. Например, его можно применять для передачи входных данных для: сетевых служб (таких как телеконференции); программ с интерфейсом в виде командной строки; аннотирования документов на сервере; выполнения операций в базах данных.

Данные, посылаемые на сервер, находятся в теле содержимого запроса клиента. По завершении обработки запроса POST и заголовков сервер передает тело содержимого в программу, заданную URL. В качестве схемы кодирования с методом POST используется Base64-кодирование, которое позволяет преобразовывать данные форм в список переменных и значений для CGI-обработки. Ниже приведен небольшой пример запроса клиента с использованием метода POST. Клиент посылает на сервер данные о дне рождения, введенные в форму:

POST /cgi-bin/birthday.pl HTTP/1.0
User-Agent; Mozilla/4.05 (WinNT; 1)
Accept: image/gif, iinage/x-xbj.tmap, image/jpeg, J.mage/pjpeg, */*
Host: **www.ora.com**
Content-type: application/x-www-form-ur.lencoded
Content-Length: 20
nionth=august&date=24

Ответы сервера

Ответ сервера на запрос клиента состоит из трех частей. Первая строка – это строка ответа сервера, которая содержит номер версии HTTP, число, обозначающее состояние запроса, и краткое описание состояния. После строки ответа следует информация заголовка и тело содержимого, если таковое имеется.

Протокол SMTP

Основная задача протокола SMTP (Simple Mail Transfer Protocol) заключается в том, чтобы обеспечивать передачу электронных сообщений (почту). Для работы через протокол SMTP клиент создаёт TCP соединение с сервером через порт 25. Затем клиент и SMTP сервер обмениваются информацией пока соединение не будет закрыто или прервано. Основной процедурой в SMTP является передача почты (Mail Procedure). Далее идут процедуры форвардинга почты (Mail Forwarding), проверка имён почтового ящика и вывод списков почтовых групп. Самой первой процедурой является открытие канала передачи, а последней – его закрытие.

При этом отправитель инициирует соединение и посылает запросы на обслуживание, а получатель – отвечает на эти запросы. Фактически отправитель выступает в роли клиента, а получатель – сервера. Канал связи устанавливается непосредственно между отправителем и получателем сообщения. При таком взаимодействии почта достигает абонента в течение нескольких секунд после отправки

Команды SMTP указывают серверу, какую операцию хочет произвести клиент. Команды состоят из ключевых слов, за которыми следует один или более параметров. Ключевое слово состоит из 4-х символов и разделено от аргумента одним или несколькими пробелами. Каждая командная строка заканчивается символами перевода строки (CRLF). Вот синтаксис всех команд протокола SMTP (SP – пробел):

HELO <SP> <domain> <CRLF>
MAIL <SP> FROM:<reverse-path> <CRLF>
RCPT <SP> TO:<forward-path> <CRLF>
DATA <CRLF>
RSET <CRLF>
SEND <SP> FROM:<reverse-path> <CRLF>
SOML <SP> FROM:<reverse-path> <CRLF>
SAML <SP> FROM:<reverse-path> <CRLF>
VRFY <SP> <string> <CRLF>
EXPN <SP> <string> <CRLF>
HELP <SP> <string> <CRLF>
NOOP <CRLF>
QUIT <CRLF>

Обычный ответ SMTP сервера состоит из номера ответа, за которым через пробел следует дополнительный текст. Номер ответа служит индикатором состояния сервера.

Отправка почты

Первым делом подключаемся к SMTP серверу через порт 25. Теперь надо передать серверу команду HELLO и наш IP адрес (здесь и далее символически обозначены: C: – запрос клиента, S: – ответ сервера):

C: HELLO 195.161.101.33
S: 250 smtp.mail.ru is ready

При отправке почты передаём некоторые нужные данные (отправитель, получатель и само письмо):

C: MAIL FROM: <отправитель>
S: 250 OK

C: RCPT TO: <получатель>
S: 250 OK

указываем серверу, что будем передавать содержание письма (заголовок и тело письма)

C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
<само письмо>
передачу письма необходимо завершить символами CRLF.CRLF
S: 250 OK

Теперь завершаем работу, отправляем команду QUIT:

S: QUIT
C: 221 smtp.mail.ru is closing transmission channel

Пример:

C: From: Drozd <vasya@tut.by>
C: To: Drol <petya@mail.ru >
C: Subject: Hello
C: Hello vasya!
C: How are you?
<CRLF.CRLF>
S: 250 OK
S: QUIT
C: 221 smtp.mail.ru is closing transmission channel

Протокол POP3

POP3 – это простейший протокол для работы пользователя с содержимым своего почтового ящика. Он позволяет только забрать почту из почтового ящика сервера на рабочую станцию клиента и удалить ее из почтового ящика на сервере. Всю дальнейшую обработку почтовое сообщение проходит на компьютере клиента.

POP3-сервер не отвечает за отправку почты, он работает только как универсальный почтовый ящик для группы пользователей. Когда пользователю необходимо отправить сообщение, он должен установить соединение с каким-либо SMTP-сервером и отправить туда свое сообщение по SMTP. Этот SMTP-

сервер может быть тем же хостом, где работает POP3-сервер, а может располагаться совсем в другом месте.

POP3- сервис, как правило, 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 – <CRLF>. 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.

При открытии TCP-соединения POP3-клиентом, POP3-сервер отправляет сообщение приветствия (далее во всех примерах POP3-протокола используются следующие обозначения: C – клиент, S – сервер POP3):

S: +OK POP3 server ready

Теперь POP3-сессия находится в состоянии авторизации (AUTHORIZATION), и клиент должен зарегистрировать себя на POP3-сервере. Это может быть выполнено либо с помощью команд USER и PASS – ввод открытых пользовательского идентификатора и пароля (именно этот способ используется чаще), либо командой APOP – авторизации цифровой подписью, на базе секретного ключа. Любой POP3 -сервер должен поддерживать хотя бы один из механизмов авторизации.

Команда USER имеет следующий формат:

USER name

Аргументом (name) является строка, идентифицирующая почтовый ящик системы. Этот идентификатор должен быть уникальным в данной почтовой системе POP3-сервера. Если ответом на эту команду является строка индикатора "+OK", клиент может отправлять команду PASS – ввод пароля или QUIT – завершить сессию. Если ответом является строка "-ERR", клиент может либо повторить команду USER, либо закрыть сессию. Примеры использования команды:

C: USER frated
S: -ERR sorry, no mailbox for frated here

или

C: USER mrose
S: +OK mrose is a real hoopy frood

Команда PASS используется только после положительного ответа на команду USER:

PASS string

Аргументом команды является строка пароля данного почтового ящика. После получения команды PASS, POP3-сервер, на основании аргументов команд USER и PASS, определяет возможность доступа к заданному почтовому ящику. Если POP3-сервер ответил "+OK", это означает, что авторизация клиента прошла успешно и он может работать со своим почтовым ящиком, т.е. сессия переходит в состояние TRANSACTION. Если POP3- сервер ответил "-ERR", то либо был введен неверный пароль, либо не найден указанный почтовый ящик:

C: USER mrose
S: +OK mrose is a real hoopy frood
C: PASS secret
S: +OK mrose's maildrop has 2 messages (320 octets)

Команда закрытия POP3-сессии: QUIT - отправляется без аргументов и всегда имеет единственный ответ "+OK", например:

C: QUIT
S: +OK dewey POP3 server signing off

После того как клиент успешно прошел процедуру авторизации в POP3-сервере, и POP3-сервер "закрыл" определенный почтовый ящик только для использования данным клиентом (для тех, кто работал с базами данных, это называется EXCLUSIVE ACCESS LOCK), POP3-сессия переходит в режим TRANSACTION, и клиент может начать работу со своей почтой.

Команда STAT (без аргументов) используется для просмотра состояния текущего почтового ящика.

В ответ POP3- сервер возвращает строку, содержащую количество и общий размер в байтах сообщений, которые клиент может получить с POP3- сервера. Сообщения, помеченные на удаление, не учитываются. Формат ответа: "+OK nn mm", где nn – количество сообщений, mm – их общий объем:

C: STAT
S: +OK 2 320

Команда RETR – используется для передачи клиенту запрашиваемого сообщения:

RETR msg

Аргумент команды – номер сообщения. Если запрашиваемого сообщения нет, возвращается отрицательный индикатор "-ERR".

C: RETR 1
S: +OK 120 octets
S: <text message>
S: .

После получения, сообщение, как правило, помечается на удаление из почтового ящика, при этом используется команда DELE**:**

DELE msg

Аргумент команды: номер сообщения. Сообщения, помеченные на удаление, реально удаляются только после закрытия транзакции, на стадии UPDATE.

C: DELE 1
S: +OK message 1 deleted
ИЛИ
C: DELE 2
S: -ERR message 2 already deleted

Для проверки состояния соединения с POP3- сервером используется команда NOOP. При активном соединении ответом на нее будет положительный индикатор "+OK":

C: NOOP
S: +OK

Ниже приведена стандартная сессия работы с POP3 -протоколом.

S: <wait for connection on TCP port 110>
C: <open connection>
S: +OK POP3 server ready
C: USER mrose
S: +OK mrose is a real hoopy frood
C: PASS secret
S: +OK mrose's maildrop has 2 messages (320 octets)
C: STAT
S: +OK 2 320
C: LIST

S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends message 1>
S: .
C: DELE 1
S: +OK message 1 deleted
C: RETR 2
S: +OK 200 octets
S: <the POP3 server sends message 2>
S: .
C: DELE 2
S: +OK message 2 deleted
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <close connection>
S: <wait for next connection>

Простота протокола POP, которая послужила росту его популярности вначале, обернулась затем отсутствием гибкости и невозможности выполнять другие операции управления почтовыми ящиками. На смену POP3 пришло новое поколение протоколов работы с электронной почтой – протоколы IMAP.

Протокол FTP

FTP (RFC-959) обеспечивает файловый обмен между удаленными пользователями. Протокол FTP формировался многие годы. Первые реализации в МТИ относятся к 1971. (RFC 114 и 141). RFC 172 рассматривает протокол, ориентированный на пользователя, и предназначенный для передачи файлов между ЭВМ. Позднее в документах RFC 265 и RFC 281 протокол был усовершенствован. Заметной переделке протокол подвергся в 1973, и окончательный вид он обрел в 1985 году. Таким образом, данный протокол является одним из старейших.

Работа FTP на пользовательском уровне содержит несколько этапов:

1. Идентификация (ввод имени-идентификатора и пароля).
2. Выбор каталога.
3. Определение режима обмена (поблочный, поточный, ascii или двоичный).
4. Выполнение команд обмена (get, mget, dir, mdel, mput или put).
5. Завершение процедуры (quit или close).

Команды FTP приведены в табл. 4.1.

Таблица 4.1

Команда	Описание
ABOR	прервать предыдущую команду FTP и любую передачу данных
LIST список файлов	список файлов или директорий
PASS пароль	пароль на сервере
PORT n1,n2,n3,n4,n5,n6	IP адрес клиента (n1.n2.n3.n4) и порт (n5 x 256 + n6)
QUIT	закрыть бюджет на сервере
RETR имя файла	получить (get) файл
STOR имя файла	положить (put) файл
SYST	сервер возвращает тип системы
TYPE тип	указать тип файла: A для ASCII, I для двоичного
USER имя пользователя	имя пользователя на сервере

Команды и отклики передаются по управляющему соединению между клиентом и сервером в формате NVT ASCII. В конце каждой строки команды или отклика присутствует пара CR, LF. Команды состоят из 3 или 4 байт, а именно из заглавных ASCII символов, некоторые с необязательными аргументами. Клиент может отправить серверу более чем 30 различных FTP команд.

FTP отклики

Отклики состоят из 3-циферных значений в формате ASCII, и необязательных сообщений, которые следуют за числами. Подобное представление откликов объясняется тем, что программному обеспечению необходимо посмотреть только цифровые значения, чтобы понять, что ответил процесс, а дополнительную строку может прочитать человек. Поэтому пользователю достаточно просто прочитать сообщение (причем нет необходимости запоминать все цифровые коды откликов). Группы откликов представлены в табл. 4.2.

Таблица 4.2

Отклик	Описание
1	2
1yz	Положительный предварительный отклик. Действие началось, однако необходимо дождаться еще одного отклика перед отправкой следующей команды.
2yz	Положительный отклик о завершении. Может быть отправлена новая команда.
3yz	Положительный промежуточный отклик. Команда принята, однако необходимо отправить еще одну команду.
4yz	Временный отрицательный отклик о завершении. Требуемое действие не произошло, однако ошибка временная, поэтому команду необходимо повторить позже.
5yz	Постоянный отрицательный отклик о завершении. Команда не была воспринята и повторять ее не стоит.
x0z	Синтаксическая ошибка.

Окончание табл. 4.2

1	2
x1z	Информация.
x2z	Соединения. Отклики имеют отношение либо к управляющему, либо к соединению данных.
x3z	Аутентификация и бюджет. Отклик имеет отношение к логированию или командам, связанным с бюджетом.
x4z	Не определено.
x5z	Состояние файловой системы.

Третья цифра дает дополнительное объяснение сообщению об ошибке. Ниже приведены некоторые типичные отклики с возможными объясняющими строками.

- 125 Соединение данных уже открыто; начало передачи.
- 200 Команда исполнена.
- 214 Сообщение о помощи (для пользователя).
- 331 Имя пользователя принято, требуется пароль.
- 425 Невозможно открыть соединение данных.
- 452 Ошибка записи файла.
- 500 Синтаксическая ошибка (неизвестная команда).
- 501 Синтаксическая ошибка (неверные аргументы).
- 502 Нереализованный тип MODE.

Управление соединением

Использовать соединение данных можно тремя способами.

1. Отправка файлов от клиента к серверу.
2. Отправка файлов от сервера к клиенту.
3. Отправка списка файлов или директорий от сервера к клиенту.

Задания к лабораторной работе № 4

Все программы ко всем заданиям выполняются на любом языке программирования, допускается использование компонент ClientSocket, ServerSocket, TcpClient, TcpServer в C++Builder/Delphi и аналогичных классов в C++/C#.

Вариант 1

Написать программу, реализующую функции HTTP-сервера версии 1.0. В обязательном порядке должны поддерживаться следующие виды запросов: GET, POST, HEAD, а так же наиболее распространенные коды ответов. Сервер конфигурируется на определенный каталог, где расположены html-файлы и другие подкаталоги. В главном окне сервера расположено поле типа memo, в котором отображается весь протокол общения HTTP клиента с HTTP сервером, например:

GET /index.html HTTP/1.O

Connection: Keep-Alive

User-Agent: Mozilla/4.05 (WinNT; 1)

Host: www.ora.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
HTTP/1.0 200 Document follows
Date: Fri, 20 Jan 1998 08:17:58 GMT
Server: Apache/1.2.6
Last-modified: Mon, 20 Jun 1997 21:53:08 GMT
Content-type: text/html
Content-length: 2482
<html><body>.....

Тестирование и подача программы-сервера производится при помощи Web-броузера.

Вариант 2

Написать программу, реализующую функции HTTP-клиента версии 1.0. В обязательном порядке должны поддерживаться следующие виды запросов: GET, POST, HEAD, а так же наиболее распространенные коды ответов. Отображение полученных данных в форматированном виде не обязательно (можно в виде plain text). В окне клиента должно быть расположено поле типа memo в котором отображается весь протокол общения HTTP клиента с HTTP сервером (см. задание 1). Тестирование и подача HTTP клиента производится при помощи запроса к реальному web-серверу, расположенному в Internet или установленному в локальной сети, или при помощи запроса к web-серверу, написанному в предыдущем задании.

Вариант 3

Написать программу, реализующую функции POP3-сервера. В главном окне сервера расположено поле типа memo, в котором отображается весь протокол общения клиента с сервером, например:

STAT
+OK 2 320
LIST
+OK 2 messages (320 octets)

Тестирование и подача программы-сервера производится при помощи любого стандартного почтового клиента.

Вариант 4

Написать программу, реализующую функции POP3-клиента. В главном окне клиента расположено поле типа memo, в котором отображается весь протокол общения клиента с сервером (см. задание 3).

Тестирование и подача программы-клиента производится при помощи любого стандартного почтового сервера, расположенного в сети Internet или локальной сети. В качестве сервера может использоваться программа, написанная к заданию 3.

Вариант 5

Написать программу, реализующую функции SMTP-сервера. В главном окне сервера расположено поле типа memo, в котором отображается весь протокол общения клиента с сервером, например:

MAIL From <Alex@mail.ru>
250 Sender ok
RCPT To:<ysemenov@mail.ch>
250 <ysemenov@mail.ch> Recepient ok
DATA

Тестирование и подача программы-сервера производится при помощи любого стандартного почтового клиента.

Вариант 6

Написать программу, реализующую функции SMTP-клиента. В главном окне клиента расположено поле типа memo, в котором отображается весь протокол общения клиента с сервером (см. задание 5).

Тестирование и подача программы-клиента производится при помощи любого стандартного почтового сервера, расположенного в сети Internet или локальной сети. В качестве сервера может использоваться программа, написанная к заданию 5.

Вариант 7

Написать программу, реализующую функции FTP-сервера. В главном окне сервера расположено поле типа memo, в котором отображается весь протокол общения клиента с сервером, например:

USER vasilisa
331 Password required for vasilisa.
PASS abcd
230 User vasilisa logged in.
PORT 140,252,13,34,4,150
200 PORT command successful.
…

Тестирование и подача программы-сервера производится при помощи любого стандартного FTP клиента.

Вариант 8

Написать программу, реализующую функции FTP-клиента. В главном окне клиента расположено поле типа memo, в котором отображается весь протокол общения клиента с сервером (см. задание 7).

Тестирование и подача программы-клиента производится при помощи любого стандартного FTP сервера, расположенного в сети Internet или локальной сети. В качестве сервера может использоваться программа, написанная к заданию 7.

Литература

К лабораторной работе №1

1. MSDN Library. Раздел Networking and Directory Services – Network Protocols – NetBIOS, раздел Networking and Directory Services – Network Management – Windows Networking (WNet).

2. Справочник по командам Windows. [Электрон, ресурс] / Режим доступа: http://winchanger.whatis.ru/file/prog.zip

3. Windows: Сети: Изучение TCP/IP. [Электрон, ресурс] / OS Zone. Режим доступа: http://www.oszone.net/display.php?id=1409

4. К.Закер. Компьютерные сети. Модернизация и поиск неисправностей: Пер. с англ. – СПб.: БХВ-Петербург, 2004. – 1008 с.: ил.

К лабораторной работе № 2

5. MSDN Library. Раздел Networking and Directory Services – Network Protocols – Windows Sockets.

6. Windows: Сети: Изучение TCP/IP: . [Электрон, ресурс] / OS Zone. Режим доступа: http://www.oszone.net/display.php?id=1409

7. К.Закер. Компьютерные сети. Модернизация и поиск неисправностей: Пер. с англ. – СПб.: БХВ-Петербург, 2004. – 1008 с.: ил.

К лабораторной работе № 3

8. [Электрон, ресурс] / Режим доступа: http://www.rfc-editor.org/rfc/

   a. RFC 854 - Telnet
   b. RFC 862 - Echo
   c. RFC 867 - DayTime
   d. RFC 868 - Time
   e. RFC 954 - WhoIs
   f. RFC 1282 - RLogin
   g. RFC 1288 - Finger

К лабораторной работе № 4

9. Паркер Т., Сиян К. TCP/IP. Для профессионалов.: Пер. с англ. – СПб.: Питер, 2003. 864 с.: ил.

10. Протоколы: HTTP, FTP, RPC, SMTP, POP3, IMAP4, SNMP, IPX/SPX - Книга. [Электрон, ресурс] / Исходники.RU. Режим доступа: http://ishodniki.ru/list/?show=protocols

11. Сетевые протоколы и технологии. [Электрон, ресурс] / OS Zone. Режим доступа: http://www.oszone.net/display.php?id=83

12. Request for comment. [Электрон, ресурс] / Режим доступа: http://www.rfc-editor.org/rfc/

Св. план 2006, поз. 55

Учебное издание

Сурков Дмитрий Андреевич
Коростель Сергей Владимирович
Мельникова Елена Владимировна
Марина Ирина Михайловна

Сети ЭВМ

Лабораторный практикум для студентов специальности I-40 01 01
«Программное обеспечение информационных технологий»
дневной формы обучения

В 2-х частях

Часть 1

Ответственный за выпуск И.М. Марина

Подписано в печать 16.06.2006. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 2,21.
Уч.-изд. л. 1,8. Тираж 100 экз. Заказ 291.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №20330/0131666 от 30.04.2004.
220013, Минск, П.Бровки, 6.