МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УО ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ ДЛЯ ПОДГОТОВКИ И ПРАКТИЧЕСКИХ ЗАНЯТИЙ

по дисциплине

«ГЛОБАЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СЕТИ»

для студентов-заочников специальностей

1-40 02 01 «Вычислительные машины, системы и сети»

Составитель: Бровко Н.В.

2013

Содержание

1. ФИО и контактная информация о преподавателе

Бровко Надежда Валерьевна – ассистент кафедры «Вычислительные системы и сети»,

ауд. 156В к/к., e-mail: nadzeya.brouka@gmail.com

2. Структура дисциплины

Целью изучения дисциплины является приобретение современных систематизированных знаний по архитектуре и принципам работы глобальных вычислительных сетей, основам их проектирования, методам анализа и синтеза, особенностям программного обеспечения.

Главной **задачей** курса является приобретение студентами знаний о видах, организации, основах функционирования и применении глобальных вычислительных сетей, а также способностей к самостоятельной разработке сетевых приложений, взаимодействующих при помощи соответствующих сетевых протоколов.

В результате изучения дисциплины студенты **должны знать**:

- основные виды архитектур вычислительных сетей различного назначения;
- принципы построения, работу и взаимодействие основных устройств названных систем в процессе их функционирования;
- назначение и принципы функционирования основных сетевых протоколов передачи данных;

Основой для изучения курса являются дисциплины: «Системное программное обеспечение», «Локальные вычислительные сети».

3. Примеры программ

В ходе самостоятельной работы необходимо научиться разрабатывать следующие приложения: клиент-серверное приложение, реализующее TCP взаимодействие с помощью механизма сокетов; клиент-серверное приложение, реализующее широковещательную рассылку UDP пакетов с помощью механизма сокетов.

Библиотеки, среды и языки программирования: Winsock2, MS Visual Studio и др., C/C++, C#.

Пример 1: клиент-серверное приложение, реализующее TCP взаимодействие с помощью механизма сокетов.

Протокол TCP (Transmission Control Protocol, Протокол контроля передачи) обеспечивает сквозную доставку данных между прикладными процессами, запущенными на узлах, взаимодействующих по сети.

TCP - надежный байт-ориентированный (byte-stream) протокол с установлением соединения. TCP находится на транспортном уровне стека TCP/IP, между протоколом IP и

собственно приложением. Протокол IP занимается пересылкой дейтаграмм по сети, никак не гарантируя доставку, целостность, порядок прибытия информации и готовность получателя к приему данных; все эти задачи возложены на протокол TCP.
При получении дейтаграммы, в поле Protocol которой указан код протокола TCP, модуль IP передает данные этой дейтаграммы модулю TCP. Эти данные представляют собой TCP-сегмент, содержащий TCP-заголовок и данные пользователя (прикладного процесса). Модуль TCP анализирует служебную информацию заголовка, определяет, какому именно процессу предназначены данные пользователя, проверяет целостность и порядок прихода данных и подтверждает их прием другой стороне. По мере получения правильной последовательности неискаженных данных пользователя они передаются прикладному процессу.
Socket (гнездо, разъем) - абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки канала связи с коммуникационной средой, образованной вычислительной сетью. При использовании протоколов TCP/IP можно говорить, что socket является средством подключения прикладной программы к порту локального узла сети. Socket-интерфейс представляет собой просто набор системных вызовов и/или библиотечных функций языка программирования СИ. Ниже рассматривается подмножество функций socket-интерфейса, достаточное для написания сетевых приложений, реализующих модель "клиент-сервер" в режиме с установлением соединения. Все функции сокетов содержаться в wsock32.dll перед тем как написать программу с использование функций сокетов в VisualC++ необходимо задать компилятору чтобы он включил в программу файл wsock32.lib. В меню Project выберите пункт settings а там укажите раздел Link, wsock32.lib можно ввести в поле Library modules или project options. Не забудьте поставить #include "Winsock2.h".

Создание сервера

Прежде чем воспользоваться функцией socket необходимо проинициализировать процесс библиотеки wsock32.dll вызвав функцию WSAStartup например:

```
WSADATA WsaData;
int err = WSAStartup (0x0101, &WsaData);
if (err == SOCKET_ERROR)
{
        printf ("WSAStartup() failed: %ld\n", GetLastError ());
return 1;
}
```

Здесь 0x0101 версия библиотеки которую следует использовать.

Теперь объявление переменную типа SOCKET например s :

```
s = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
```

Создание socket'a осуществляется следующим системным вызовом

```
int socket (domain, type, protocol)
int domain;
int type;
int protocol;
```

Аргумент *domain* задает используемый для взаимодействия набор протоколов (вид коммуникационной области), для стека протоколов TCP/IP он должен иметь символьное значение AF_INET.
Аргумент *type* задает режим взаимодействия:

- SOCK_STREAM - с установлением соединения;
- SOCK_DGRAM - без установления соединения.

Аргумент *protocol*задает конкретный протокол транспортного уровня (из нескольких возможных в стеке протоколов). Если этот аргумент задан равным 0, то будет использован протокол "по умолчанию" (TCP для SOCK_STREAM и UDP для SOCK_DGRAM при использовании комплекта протоколов TCP/IP).
При удачном завершении своей работы данная функция возвращает дескриптор socket'a - целое неотрицательное число, однозначно его идентифицирующее. Дескриптор socket'a аналогичен дескриптору файла ОС UNIX.
При обнаружении ошибки в ходе своей работы функция возвращает число "-1".
Далее мы задаем параметры для сокета (сервера) для этого нам необходимо объявить структуру SOCKADDR_IN sin далее заполняем параметры для сервера:

```
SOCKADDR_IN sin;
sin.sin_family = AF_INET;
sin.sin_port = htons(80);
sin.sin_addr.s_addr = INADDR_ANY;
```

Структура SOCKADDR_IN используется несколькими системными вызовами и функциями socket-интерфейса и определена в include-файле in.h следующим образом:

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

Поле sin_family определяет используемый формат адреса (набор протоколов), в нашем случае (для TCP/IP) оно должно иметь значение AF_INET.
Поле sin_addr содержит адрес (номер) узла сети.
Поле sin_port содержит номер порта на узле сети.
Поле sin_zero не используется.
Определение структуры in_addr (из того же include-файла) таково:

```
struct in_addr
{
        union
        {
        u_long S_addr;
        /*
        другие (не интересующие нас)
        члены объединения
```

```
        */
        } S_un;
        #define s_addr S_un.S_addr
    };
```

Структура SOCKADDR_IN должна быть полностью заполнена перед выдачей системного вызова bind. При этом, если поле sin_addr.s_addr имеет значение INADDR_ANY, то системный вызов будет привязывать к socket'у номер (адрес) локального узла сети.

Для подключения socket'а к коммуникационной среде, образованной вычислительной сетью, необходимо выполнить системный вызов bind, определяющий в принятом для сети формате локальный адрес канала связи со средой. В сетях TCP/IP socket связывается с локальным портом. Системный вызов bind имеет следующий синтаксис:

```
 int bind (s, addr, addrlen)
    int s;
    struct SOCKADDR_IN *addr;
    int addrlen;
```

Пример:

```
        err = bind( s, (LPSOCKADDR)&sin, sizeof(sin) );
```

Аргумент *s* задает дескриптор связываемого socket'а.

Аргумент *addr* в общем случае должен указывать на структуру данных, содержащую локальный адрес, приписываемый socket'у. Для сетей TCP/IP такой структурой является SOCKADDR_IN.

Аргумент *addrlen* задает размер (в байтах) структуры данных, указываемой аргументом addr.

В случае успеха bind возвращает 0, в противном случае - "-1".

Для установления связи "клиент-сервер" используются системные вызовы listen и accept (на стороне сервера), а также connect (на стороне клиента). Для заполнения полей структуры socaddr_in, используемой в вызове connect, обычно используется библиотечная функция gethostbyname, транслирующая символическое имя узла сети в его номер (адрес).

Системный вызов listen выражает желание выдавшей его программы-сервера ожидать запросы к ней от программ-клиентов и имеет следующий вид:

```
        int listen (s, n)
    int s;
    int n;
```

Пример:

```
        err = listen( s, SOMAXCONN);
```

Аргумент *s* задает дескриптор socket'а, через который программа будет ожидать запросы к ней от клиентов. Socket должен быть предварительно создан системным вызовом socket и обеспечен адресом с помощью системного вызова bind.

Аргумент *n* определяет максимальную длину очереди входящих запросов на установление связи. Если какой-либо клиент выдаст запрос на установление связи при полной очереди, то этот запрос будет отвергнут.

Признаком удачного завершения системного вызова listen служит нулевой код возврата.
Перед тем как воспользоваться функцией accept сначала объявите ещё одну переменную типа SOCKET, например s1.

```
SOCKADDR_IN from;
int fromlen=sizeof(from);
s1 = accept(s,(struct sockaddr*)&from, &fromlen);
```

Это сделано для того что бы узнать IP адрес и порт удаленного компьютера. Что бы вывести на экран IP адрес и порт удаленного компьютера можно воспользоваться следующим кодом:

```
printf("accepted connection from %s, port %d\n", inet_ntoa(from.sin_addr),
htons(from.sin_port)) ;
```

Для приема запросов от программ-клиентов на установление связи в программах-серверах используется системный вызов accept, имеющий следующий вид:

```
int accept (s, addr, p_addrlen)
int s;
struct sockaddr_in *addr;
int *p_addrlen;
```

Аргумент *s* задает дескриптор socket'a, через который программа-сервер получила запрос на соединение (посредством системного запроса listen).
Аргумент *addr* должен указывать на область памяти, размер которой позволял бы разместить в ней структуру данных, содержащую адрес socket'a программы-клиента, сделавшей запрос на соединение. Никакой инициализации этой области не требуется.
Аргумент *p_addrlen* должен указывать на область памяти в виде целого числа, задающего размер (в байтах) области памяти, указываемой аргументом addr.
Системный вызов accept извлекает из очереди, организованной системным вызовом listen, первый запрос на соединение и возвращает дескриптор нового (автоматически созданного) socket'a с теми же свойствами, что и socket, задаваемый аргументом s. Этот новый дескриптор необходимо использовать во всех последующих операциях обмена данными.
Кроме того после удачного завершения accept:

1. область памяти, указываемая аргументом addr, будет содержать структуру данных (для сетей TCP/IP это sockaddr_in), описывающую адрес socket'a программы-клиента, через который она сделала свой запрос на соединение;
2. целое число, на которое указывает аргумент p_addrlen, будет равно размеру этой структуры данных.

Если очередь запросов на момент выполнения accept пуста, то программа переходит в состояние ожидания поступления запросов от клиентов на неопределенное время (хотя такое поведение accept можно и изменить).
Признаком неудачного завершения accept служит отрицательное возвращенное значение (дескриптор socket'a отрицательным быть не может).
Примечание. *Системный вызов* accept *используется в программах-серверах, функционирующих только в режиме с установлением соединения.*
После соединения с клиентом для передачи информации используются команды send и:

```
BYTE RecvBuffer[1];
while(recv(s1,RecvBuffer,sizeof(RecvBuffer),0)!=SOCKET_ERROR)
{
printf("%c",RecvBuffer[0]);
send(s1,MsgText,sizeof(MsgText),MSG_DONTROUTE);
}
```

Цикл while будет выполнятся пока клиент не отключится

Для получения данных от партнера по сетевому взаимодействию используется системный вызов recv, имеющий следующий вид

```
int recv (s, buf, len, flags)
int s;
char *buf;
int len;
int flags;
```

Аргумент *s* задает дескриптор socket'a, через который принимаются данные.

Аргумент *buf* указывает на область памяти, предназначенную для размещения принимаемых данных.

Аргумент *len* задает длину (в байтах) этой области.

Аргумент *flags* модифицирует исполнение системного вызова recv. При нулевом значении этого аргумента вызов recv полностью аналогичен системному вызову read.

При успешном завершении recv возвращает количество принятых в область, указанную аргументом *buf*, байт данных. Если канал данных, определяемый дескриптором s, оказывается "пустым", то recv переводит программу в состояние ожидания до момента появления в нем данных.

Для посылки данных партнеру по сетевому взаимодействию используется системный вызов send, имеющий следующий вид

```
int send (s, buf, len, flags)
int s;
char *buf;
int len;
int flags;
```

Аргумент *s* задает дескриптор socket'a, через который посылаются данные.

Аргумент *buf* указывает на область памяти, содержащую передаваемые данные.

Аргумент *len* задает длину (в байтах) передаваемых данных.

Аргумент *flags* модифицирует исполнение системного вызова send. При нулевом значении этого аргумента вызов send полностью аналогичен системному вызову write.

При успешном завершении send возвращает количество переданных из области, указанной аргументом buf, байт данных. Если канал данных, определяемый дескриптором s, оказывается "переполненным", то send переводит программу в состояние ожидания до момента его освобождения.

Для закрытия ранее созданного socket'a используется обычный системный вызов closesocket, применяемый в ОС UNIX для закрытия ранее открытых файлов и имеющий следующий вид

int closesocket(s)
int s;
Аргумент *s* задает дескриптор ранее созданного socket'a.
Однако в режиме с установлением логического соединения (обеспечивающем, как правило, надежную доставку данных) внутрисистемные механизмы обмена будут пытаться передать/принять данные, оставшиеся в канале передачи на момент закрытия socket'a. На это может потребоваться значительный интервал времени, неприемлемый для некоторых приложений. В такой ситуации необходимо использовать описываемый далее системный вызов **shutdown**.

Создание клиента

Программа клиента делается аналогично до момента создания сокетов. Создайте сокет так как описано выше, но не пользуйтесь командой bind:

```
SOCKADDR_IN anAddr;
anAddr.sin_family = AF_INET;
anAddr.sin_port = htons(80);
anAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
```

Заполнение структуры производится почти также но нужно указать теперь IP адрес сервера (пример 127.0.0.1) .Дальше сразу можно соединятся:

```
connect(s, (struct sockaddr *)&anAddr, sizeof(struct sockaddr));
```

Для обращения программы-клиента к серверу с запросом на установление логической соединения используется системный вызов connect, имеющий следующий вид

```
int connect (s, addr, addrlen)
int s;
struct sockaddr_in *addr;
int addrlen;
```

Аргумент *s* задает дескриптор socket'a, через который программа обращается к серверу с запросом на соединение. Socket должен быть предварительно создан системным вызовом socket и обеспечен адресом с помощью системного вызова bind.
Аргумент *addr* должен указывать на структуру данных, содержащую адрес, приписанный socket'у программы-сервера, к которой делается запрос на соединение. Для сетей TCP/IP такой структурой является sockaddr_in. Для формирования значений полей структуры sockaddr_in удобно использовать функцию gethostbyname.
Аргумент *addrlen* задает размер (в байтах) структуры данных, указываемой аргументом addr.
Для того, чтобы запрос на соединение был успешным, необходимо, по крайней мере, чтобы программа-сервер выполнила к этому моменту системный вызов listen для socket'a с указанным адресом.
При успешном выполнении запроса системный вызов connect возвращает 0, в противном случае - "-1" (устанавливая код причины неуспеха в глобальной переменной errno).
Примечание. *Если к моменту выполнения connect используемый им socket не был привязан к адресу посредством* bind **,***то такая привязка будет выполнена автоматически.*

Вот наконец установлена долгожданная связь с сервером(не забывайте проверять ошибки). Дальше воспользуемся функциями send и recv по своему усмотрению.

Приложение

Для получения адреса узла сети TCP/IP по его символическому имени используется библиотечная функция

```
struct hostent *gethostbyname (name)
char *name;
```

Аргумент *name* задает адрес последовательности литер, образующих символическое имя узла сети.

При успешном завершении функция возвращает указатель на структуру hostent, определенную в include-файле netdb.h и имеющую следующий вид

```
struct hostent
{
    char *h_name;
    char **h_aliases;
    int h_addrtype;
    int h_lenght;
    char *h_addr;
};
```

Поле h_name указывает на официальное (основное) имя узла.

Поле h_aliases указывает на список дополнительных имен узла (синонимов), если они есть.

Поле h_addrtype содержит идентификатор используемого набора протоколов, для сетей TCP/IP это поле будет иметь значение AF_INET.

Поле h_length содержит длину адреса узла.

Поле h_addr указывает на область памяти, содержащую адрес узла в том виде, в котором его используют системные вызовы и функции socket-интерфейса.

Для "экстренного" закрытия связи с партнером (путем "сброса" еще не переданных данных) используется системный вызов shutdown, выполняемый перед close и имеющий следующий вид

```
int shutdown (s, how)
 int s;
 int how;
```

Аргумент *s* задает дескриптор ранее созданного socket'a.

Аргумент *how* задает действия, выполняемые при очистке системных буферов socket'a:

- 0 - сбросить и далее не принимать данные для чтения из socket'a;
- 1 - сбросить и далее не отправлять данные для посылки через socket;
- 2 - сбросить все данные, передаваемые через socket в любом направлении.

Пример 2: клиент-серверное приложение, реализующее широковещательную рассылку UDP пакетов с помощью механизма сокетов.

Датаграмные сокеты используют UDP, ненадежный протокол без установления соединения. UDP сервер не должен прослушивать (listen) и принимать (**accept**) клиентские

соединения, а UDP клиент не должен устанавливать соединение с сервером (**connect**). На рисунке 1 показано взаимодействие UDP сервера и UDP клиента.

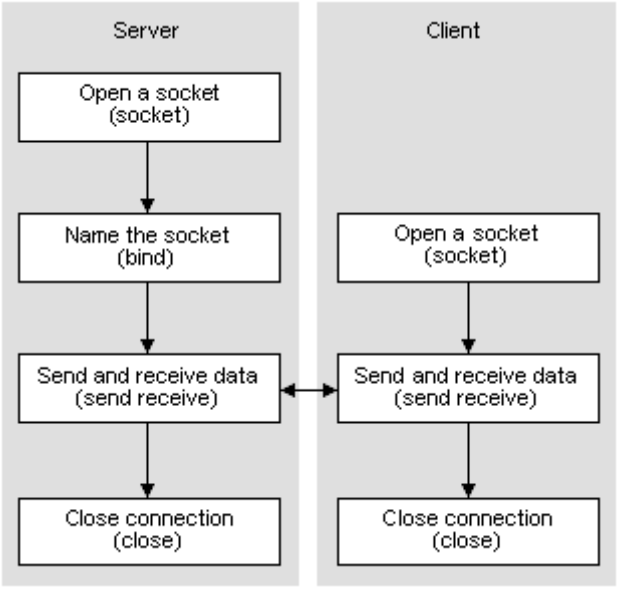

Рисунок 1 - Схема взаимодействия UDP сервера и UDP клиента

Создание сервера на базе UDP дататаграммного сокета

1. Откройте датаграмный сокет socket .
2. Используйте AF_INET в качестве формата адреса (*address format* parameter) и SOCK_DGRAM для параметра типа (*type* parameter).

```
int WinSocket;
if((WinSocket=socket(AF_INET,SOCK_DGRAM,0))==INVALID_SOCKET)
{
        printf("Allocating socket failed. Error: %d",
        WSAGetLastError());
        return -1;
}
```

3. Именуйте сокет функцией bind, используя структуру SOCKADDR_IN для параметра адреса (*address* parameter). Следующий пример реализует инициализацию (именование) сокета.

```
#define DEST_PORT 4567
SOCKADDR_IN local_sin, recv_sin;
// Fill out the local socket's address information.
local_sin.sin_family = AF_INET;
local_sin.sin_port = htons (DEST_PORT);
local_sin.sin_addr.s_addr = htonl (INADDR_ANY);
// Associate the local address with WinSocket.
if (bind (WinSocket,(struct sockaddr *)&local_sin,
sizeof (local_sin)) == SOCKET_ERROR)
{
printf ("Binding socket failed. Error: %d", WSAGetLastError ());
  closesocket (WinSocket);
  return -1;
```

```
}
```

4. Обмен данными с клиентом осуществляется функциями sendto и recvfrom. При использовании функции sendto успешная отправка пакета не означает его успешную доставку.

```
char szMessageA [100];
TCHAR szMessageW [100];
int iRecvLen , index;
iRecvLen = sizeof (recv_sin);

// Получение данных от клиента.

printf("Wait message from client.");
if(recvfrom(WinSocket,szMessageA,sizeof(szMessageA),0,
(struct sockaddr *) &recv_sin, &iRecvLen) == SOCKET_ERROR)
{
  printf("recvfrom faild! Error: %d", WSAGetLastError ());
  closesocket (WinSocket);
  return -1;
}
else
{
  // Convert the ASCII string to Unicode.
for (index = 0; index <= sizeof (szMessageA); index++)
{
  szMessageW [index] = szMessageA[index];
  // Display the string received from the client.
  printf ("Received From Client: %s", szMessageW);
}
```

5. Для закрытия сокета используйте функцию closesocket.

```
shutdown (WinSocket, 0x00);//Блокировка приема reciving)
        //на сокет WinSocket перед закрытием
closesocket(WinSocket);
WSACleanup();
```

Создание клиента на базе UDP дататаграммного сокета

1. Откройте сокет функцией socket и именуйте его функцией bind**.**

```
int WinSocket;
#define LOCAL_PORT 0
        // получить любой не занятый
if ((WinSocket = socket (AF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET)
```

```
{
  printf ("Allocating socket failed. Error: %d",     WSAGetLastError ());
  return -1;
}
SOCKADDR_IN local_sin, dest_sin;
        // Fill out the local socket's address information.
local_sin.sin_family = AF_INET;
local_sin.sin_port = htons (LOCAL_PORT);
local_sin.sin_addr.s_addr = htonl (INADDR_ANY);
        // Associate the local address with WinSocket.
if (bind (WinSocket,
        (struct sockaddr *) &local_sin,
        sizeof (local_sin)) == SOCKET_ERROR)
{
  printf ("Binding socket failed. Error: %d",    WSAGetLastError ());
  closesocket (WinSocket);
  return -1;
}
```

2. Обмен данными с сервером осуществляется при помощи функций sendto и recvfrom.

```
        // Отправка сообщения на сервер
#define DEST_PORT 4567
#define DEST_ADDR "127.0.0.1" // на локальную машину
dest_sin.sin_family = AF_INET;
dest_sin.sin_port = htons (DEST_PORT);
dest_sin.sin_addr.s_addr = inet_addr (DEST_ADDR);
char szMessageA[] = "Сообщение от клиента.";
if (sendto (WinSocket,
         szMessageA,
         strlen (szMessageA) + 1,
         0,
         (struct sockaddr FAR *) &dest_sin,
         sizeof (dest_sin)) == SOCKET_ERROR)
{
  printf ("sendto faild! Error: %d", WSAGetLastError());
  closesocket (WinSocket);
  return -1;
}
else
 printf ("Sending data succeeded!");
```

3. Закройте соединение функцией closesocket.

```
shutdown (WinSocket, 0x01); // Блокировка отправки (sending) на // сокете
WinSocket перед закрытием
```

```
closesocket (WinSocket);
WSACleanup();
```

Широковещательная рассылка UDP пакетов

Существует три типа IP адресов: персональный (unicast), широковещательный (broadcast) и групповой (multicast) .

Широковещательные и групповые запросы применимы только к UDP, подобные типы запросов позволяют приложению послать одно сообщение нескольким получателям. TCP - протокол, ориентированный на соединение, с его помощью устанавливается соединение между двумя хостами (по указанному IP адресу) с использованием одного процесса на каждом хосте (который идентифицируется по номеру порта).

Представьте себе несколько хостов в Ethernet сети. Каждый Ethernet фрейм содержит Ethernet адрес источника и назначения (48 бит). И обычно каждый фрейм предназначается одному получателю. Адрес назначения, указывающий на один интерфейс, - называется персональным (unicast). Остальные хосты, присутствующие на кабеле, не участвуют в общении между двумя хостами (если не учитывать то, что все хосты находятся все-таки на одном кабеле).

Однако иногда возникает необходимость послать фрейм всем хостам, находящимся на кабеле, - это называется широковещательной рассылкой (broadcast). Групповая адресация логически находится между персональной и широковещательной: фрейм должен быть доставлен определенному количеству хостов, которые принадлежат к группе.

Для того чтобы понять принцип широковещательной и групповой адресации, необходимо отметить, что на каждом хосте происходит фильтрация, каждый раз, когда фрейм проходит по кабелю. На рисунке 2 показано как это происходит.

Во-первых, сетевая плата просматривает каждый фрейм, который передается по кабелю, и определяет, необходимо ли принять этот фрейм и доставить его в драйвер устройства. Обычно сетевая плата принимает только те фреймы, адрес назначения которых совпадает с аппаратным адресом интерфейса или с широковещательным адресом. В дополнение, большинство интерфейсов могут находиться в смешанном режиме, когда она принимает копию каждого фрейма. Этот режим используется, например, программой tcpdump.

Рисунок 2 - Фильтрация, которая осуществляется стеком протоколов, когда принимается фрейм

В настоящее время большинство интерфейсов могут быть сконфигурированы таким образом, чтобы принимать фреймы, IP адрес которых является групповым адресом или групповым адресом какой-либо подгруппы. В групповом адресе Ethernet младший бит старшего байта установлен в единицу. В шестнадцатиричном представлении этот бит выглядит следующим образом: 01:00:00:00:00:00. (Мы можем считать, что широковещательный адрес Ethernet ff:ff:ff:ff:ff:ff это особый случай группового адреса Ethernet.)

Когда сетевая плата получает фрейм, она передает его в драйвер устройства. (Сетевая плата может отбросить фрейм только в том случае, если не сошлась контрольная сумма Ethernet.) Дополнительная фильтрация осуществляется драйвером устройства. Во-первых, тип фрейма должен принадлежать соответствующему протоколу (IP, ARP и так далее). Во-вторых, может быть осуществлена дополнительная групповая фильтрация, чтобы проверить, принадлежит ли хост к адресуемой группе.

Затем драйвер устройства передает фрейм следующему уровню, например, IP, если фрейм является IP датаграммой. IP также осуществляет фильтрацию, основанную на проверке IP адресов источника и назначения, и, в свою очередь, передает датаграмму следующему уровню (например TCP или UDP, если все в порядке).

Каждый раз когда UDP получает датаграмму от IP, он осуществляет фильтрацию, основанную на номере порта назначения, а иногда и на номере порта источника. Если указанный порт не обслуживается в текущий момент каким-либо процессом, датаграмма отбрасывается, и генерируется ICMP сообщение о недоступности порта. (TCP осуществляет подобную фильтрацию, основанную на своих номерах портов.) Если в UDP датаграмме обнаружена ошибка контрольной суммы, UDP молча ее отбрасывает.

Проблема широковещательных запросов заключается в том, что хосты, которые совсем не заинтересованы в получении этих запросов, должны их обрабатывать. Представьте себе приложение, которое должно обрабатывать широковещательные запросы UDP. Если на кабеле находится 50 хостов, но только из них 20 участвуют в работе этого приложения, в каждый момент времени один из 20 посылает

широковещательный запрос UDP, при этом остальные 30 хостов должны обработать этот запрос, который проходит весь путь по стеку протоколов до UDP уровня, прежде чем датаграмма будет отброшена. UDP датаграмма отбрасывается этими 30-ю хостами, потому что порта назначения не обслуживается каким-либо процессом.

Основная задача групповых запросов - уменьшить загрузку хостов, не учавствующих в работе определенного приложения. Хост может принадлежать к одной или нескольким группам. Если это возможно, сетевая плата сообщает, к какой группе принадлежит хост, после чего сетевой платой принимаются только фреймы определенной группы.

Широковещательную отправку пакетов можно реализовать на базе UDP протокола датаграммными сокетами.

1. Для организации широковещательной отправки необходимо перед отправкой датаграммных пакетов разрешить функцией setsockopt() для параметра SO_BROADCAST широковещательную адресацию. Пример приведен ниже по тексту.

```
// Разрешаем широковещательную (broadcast) отправку пакетов
//функцией setsockopt ().
BOOL fBroadcast = TRUE;
err = setsockopt (WinSocket,SOL_SOCKET,SO_BROADCAST,(CHAR
*)&fBroadcast,sizeof(BOOL));
if ( SOCKET_ERROR == err )
printf("setsockopt faild!  Error: %d\n",WSAGetLastError());
```

2. Отправлять пакеты на адрес INNADDR_BROADCAST (всем).

```
dest_sin.sin_family = AF_INET;
dest_sin.sin_port = htons (DEST_PORT);
dest_sin.sin_addr.s_addr = htonl (INADDR_BROADCAST);
char szMessageA[] = "\0";
lstrcpy ( szMessageA, "Message from client (Hi-Fi)." );
printf (szMessageA);
if (sendto (WinSocket,szMessageA,strlen(szMessageA)+1,0,      (struct sockaddr FAR *)
&dest_sin,sizeof(dest_sin))==SOCKET_ERROR)
{
  printf ("sendto faild! Error: %d\n", WSAGetLastError());
  closesocket (WinSocket);
  return FALSE;
}
else printf ("Sending data succeeded!\n");
```

4. Вопросы к зачету

1. Принципы объединения сетей на основе протоколов сетевого уровня.
2. Протоколы маршрутизации и функции маршрутизации.
3. Показатели и критерии алгоритмов маршрутизации.
4. Конечные системы (ES), промежуточные системы (IS), области и домены в соответствии с моделью иерархической маршрутизации стандарта OSI. Внутридоменные и междоменные протоколы маршрутизации.
5. Межсетевое взаимодействие на основе стека протоколов TCP/IP и IP-сети.
6. Доменные имена и адресация в IP-сетях. Классы IP-пакетов и выделенные адреса.
7. Отображение доменных имен на IP-адреса и система DNS.
8. Протоколы маршрутизации в IP-сетях. Формат IP пакета.
9. Протоколы разрешения адреса ARP и RARP.
10. Формат TCP-пакета и транспортный протокол TCP.
11. Формат UDP-пакета и протокол доставки дейтаграмм UDP.
12. Структура и функции глобальной сети. Интерфейсы DTE-DCE.
13. Типы глобальных сетей: выделенные каналы, сети с коммутацией каналов, сети с коммутацией пакетов, магистральные сети и сети доступа.
14. Аналоговые выделенные линии.
15. Цифровые выделенные линии.
16. Устройства DSU/CSU для подключения к выделенному каналу.
17. Протоколы канального уровня для выделенных линий: SLIP, HDLC, PPP.
18. Глобальные свези на основе аналоговых и цифровых сетей с коммутацией каналов. Технология ISDN.
19. Компьютерные глобальные сети с коммутацией пакетов. Особенности технологий X25, Frame Relay и ATM.
20. Глобальная сеть Internet. Протоколы Internet: FTP, http, SNMP, SMTP.

5. Список литературы

Учебники доступны в электронном виде по ссылке:
https://docs.google.com/folder/d/0B3ToAEp6FEQ2OXFoOWQ4a0szWXM/edit?usp=sharing

1. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов, 4-е изд. – СПб.: Питер, 2010.
2. Анкудинов Г. И., Анкудинов И. Г., Стрижаченко А. И. Сети ЭВМ и телекоммуникации. Архитектура и сетевые технологии: Учеб. Пособие. – [Новое изд.]. – СПб.: СЗТУ, 2006.