**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ГРАЖДАНСКОЙ АВИАЦИИ**

Н.И. Черкасова, Л.А. Надейкина

ПРОГРАММИРОВАНИЕ СЕТЕВЫХ ПРИЛОЖЕНИЙ

ПОСОБИЕ

по выполнению лабораторной работы

для студентов IV курса
направления 09.03.01
очной формы обучения

Москва-2016

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)**

Кафедра вычислительных машин, комплексов, систем и сетей

Н.И. Черкасова, Л.А. Надейкина

ПРОГРАММИРОВАНИЕ СЕТЕВЫХ ПРИЛОЖЕНИЙ

ПОСОБИЕ

по выполнению лабораторной работы

для студентов IV курса
направления 09.03.01
очной формы обучения

Москва-2016

ББК 6ф7.3
Ч 48

Рецензент канд. техн. наук, доц. Н.И. Романчева

Черкасова Н.И., Надейкина Л.А.

Ч 48 Программирование сетевых приложений: пособие по выполнению лабораторной работы. – М.: МГТУ ГА, 2016. – 24 с.

Данное пособие издается в соответствии с рабочей программой учебной дисциплины «Программирование сетевых приложений» по учебному плану для студентов IV курса направления 09.03.01 очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 25.10.2016 г. и методического совета 25.10.2016 г.

Подписано в печать 15.11.2016 г.		
Печать офсетная	Формат 60x84/16	1,10 уч.-изд. л.
1,39 усл.печ.л.	Заказ № 123	Тираж 60 экз.

Московский государственный технический университет ГА
125993 Москва, Кронштадтский бульвар, д.20
Редакционно-издательские услуги ООО «Имидж-студия Арина»
127051 Москва, М. Сухаревская пл., д. 2/4 стр.1

Содержание

1.Цель лабораторной работы

Целью лабораторной работы является освоение:

1. Особенностей многоадресной рассылки и широковещания.
2. Приемов работы с использованием интерфейса программирования Windows Sockets для различных сетевых протоколов.
3. Структуры и состава интерфейса программирования Windows Sockets.
4. Составления приложений, поддерживающих многоадресную рассылку и широковещание с использованием интерфейса программирования Windows Sockets.

2.Содержание отчёта

Отчет по лабораторной работе должен включать:

1) цель лабораторной работы;
2) конкретный вариант задания на выполнение;
3) тексты программ;
4) схемы алгоритмов;
5) результаты выполнения программ.

3.Задание на выполнение:

3.1.Создать приложение, ведущее рассылку сообщений с использованием механизма многоадресной рассылки.

3.2.Создать приложение, дейтаграммный клиент и сервер winsocket (широковещание).

4.Теретические сведения

4.1.Интерфейс прикладного программирования Windows Sockets (Winsock)

Windows Sockets (Winsock) – это сетевой интерфейс прикладного программирования, а не протокол, основной интерфейс доступа к различным сетевым базовым протоколам, реализованный на всех платформах. Winsock многое унаследовал от реализации BSD Sockets ,но также включает Microsoft-specific расширения, постоянно продолжая развиваться и расширяться. Winsock поддерживает надежное, ориентированное на соединение, а также ненадежное, не ориентированное

на соединение взаимодействия. Windows 2000 включает Winsock 2.2, который доступен для потребительских версий в виде надстройки.
Winsock обеспечивает:
1. поддержку по механизму scatter-gather и ассинхронный ввод-вывод I/O.
2. поддержку Quality of service (QoS) если нижележащая сеть поддерживает QoS, приложения могут согласовывать между собой максимальные задержки и полосы пропускания.
3. расширяемость - Winsock можно использовать не только с протоколами, поддерживаемыми Windows 2000 но и с другими.
4. поддержку интегрированных пространств имен, отличных от определенных протоколом, который использует приложение вместе с Winsock. Например, сервер может опубликовать свое имя в Active Directory, а клиент, используя расширения пространства имен найти адрес сервера в Active Directory.
5. поддержку многоадресных сообщений передаваемых сразу нескольким адресатам.

Функционирование Winsock.

Для создания Winsock-приложения необходимо создать сокет, являющийся конечной точкой коммуникационного соединения. Сокет необходимо привязать к адресу конкретного локального компьютера. Поскольку сокет не зависит от протокола, можно выбрать любой протокол, установленный в системе, где работает сокет. Далее схемы работы различных типов сокетов отличаются. Winsock-сервер, ориентированный на логическое соединение, выполняет на сокете операцию listen, указывая число поддерживаемых соединений. Далее выполняется операция accept, которая при наличии ждущего запроса на соединение завершается немедленно, в противном случае после поступления запроса. После установления соединения accept возвращает новый сокет с серверной стороны. Сервер может выполнять операции передачи и приема данных.

Клиенты подключаются к серверу с помощью вызова функции connect с указанием адреса удаленного компьютера. После установления соединения клиент может принимать и посылать сообщения через свой сокет. На рисунке представлена коммуникационная связь между клиентом и сервером, ориентированными на логическое соединение.

После привязки к адресу сервер, не требующих логических соединений, ничем не отличается от аналогичного клиента. Сервер посылает и получает сообщение через сокет, просто указывая требуемый адрес. Отправитель получает код ошибки в ходе следующей операции приема, если предыдущее сообщение не было доставлено.

4.2.Механизм широковещания.

Под широковещанием предполагается передача данных от одной рабочей станции всем рабочим станциям ЛВС. Все компьютеры сети могут получать и обрабатывать подобные сообщения, если они поддерживают неориентированные на соединения протоколы.

Высокая рабочая нагрузка сети при широковещании, является следствием требования обработки широковещательного сообщения каждого компьютера ЛВС, хотя большинству компьютеров сети эти данные не нужны и они отбрасываются. Тем не менее время на обработку пакетов данных тратится.

Существуют три типа IP адресов: персональный (unicast), широковещательный (broadcast) и групповой (multicast). Рассмотрим широковещательные адреса более подробно.

Каждый раз, когда UDP получает дейтаграмму 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, дейтаграмма отбрасывается, и генерируется ICMP сообщение о недоступности порта. (TCP осуществляет подобную фильтрацию, основанную на своих номерах портов.) Если в UDP дейтаграмме обнаружена ошибка контрольной суммы, UDP ее отбрасывает.

Проблема широковещательных запросов заключается в том, что хосты, которые совсем не заинтересованы в получении этих запросов, должны их обрабатывать. Если в сети находится 40 хостов, но только 10 из них участвуют в работе этого приложения, в каждый момент времени один из 10 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 дейтаграмма будет отброшена. UDP дейтаграмма отбрасывается этими 30 хостами.

Широковещательные запросы

Рассмотрим различные формы широковещательных адресов IP.
Ограниченный широковещательный запрос

Ограниченный широковещательный адрес (limited broadcast address) - это адрес 255.255.255.255. Он может быть использован в качестве адреса

назначения для IP дейтаграмм в процессе конфигурации хоста, когда хост может не знать собственной маски подсети и даже своего IP адреса.
Дейтаграмма, направляющаяся на ограниченный широковещательный адрес, никогда (ни при каких условиях) не будет перенаправлена маршрутизатором. Она может существовать только на локальном кабеле.

При наличии нескольких интерфейсов хоста, большинство систем воспринимают 255.255.255.255 как адрес широковещательного адреса первого интерфейса, который был сконфигурирован, и не позволяют послать дейтаграмму на все подключенные интерфейсы, поддерживающие широковещательную адресацию.

Широковещательный запрос в сеть

В широковещательном адресе сети (net-directed broadcast) идентификатор хоста устанавливается во все единичные биты. Широковещательный адрес для сети класса А, имеет вид netid.255.255.255, где netid это идентификатор сети класса А.

Маршрутизаторы должны перенаправлять широковещательные запросы, направляемые в сеть, при этом, однако, должна присутствовать опция, позволяющая отключить это перенаправление.

Широковещательный запрос в подсеть

Широковещательный адрес подсети (subnet-directed broadcast address) имеет идентификатор хоста, установленный в единицы, однако определенный идентификатор подсети. Для того, чтобы классифицировать адрес как широковещательный адрес подсети, необходимо знать маску подсети.

Широковещательный запрос, направленный во все подсети

Широковещательный адрес всех подсетей (all-subnets-directed broadcast address), также требует знания маски подсети в сети назначения. Только в этом случае можно определить различие между широковещательным адресом всех подсетей и широковещательным адресом сети. И идентификатор подсети, и идентификатор хоста, в данном случае, устанавливаются в единицы. Например, если маска подсети назначения 255.255.255.0, то IP адрес 128.1.255.255 это широковещательный запрос, направленный во все подсети. Однако, если сеть не разбита на подсети, это широковещательный запрос, направляемый в сеть.

Но такой тип широковещательных запросов является устаревшим , и используются групповые запросы, а не широковещательные запросы, направленные во все подсети.

4.3. Многоадресное вещание.

Способность одного процесса передавать данные одному или более получателям называется многоадресная рассылка. Однако, методика присоединения процесса к многоадресному сеансу зависит от применяемого для передачи данных протокола.

Многоадресная передача по протоколу IP является видоизмененной формой широковещания. Для нее необходимо, чтобы все заинтересованные в приеме и передаче узлы были членами особой группы.

При присоединении группы к многоадресному вещанию, на сетевом адаптере добавляется фильтр. Он требует от сетевого оборудования обрабатывать и транслировать по сетевому стеку до соответствующего процесса только данные, предназначенные групповому адресу, к которому присоединился процесс.

Многоадресная рассылка

Технология многоадресной рассылки (multicasting) позволяет отправлять данные от одной рабочей станции в сети, а затем тиражировать их для остальных, не создавая большой нагрузки на сеть. Она является альтернативой широковещанию (broadcasting). При многоадресной рассылке данные передаются в сеть, только если их запрашивает получатель.

На платформе Win32 протоколов, поддерживающих многоадресную рассылку и доступных из Winsock, только два: IP и ATM. Отметим, что некоторые устаревшие сетевые адаптеры не могут принимать и отправлять информацию по адресам многоадресной IP-рассылки.

Многоадресная рассылка обладает двумя важными характеристиками: плоскостью управления (control plane) и плоскостью данных (data plane).
Первая определяет способ организации членства в группах, вторая - способ распространения данных среди членов группы. Любая из этих плоскостей может быть корневой или равноправной. Пример корневой плоскости управления – протокол ATM.

Равноправная плоскость управления позволяет соединиться с группой любому участнику. При этом, протокол реализованный для членов группы зависит от программиста. Равноправной плоскостью управления, например, является многоадресная IP-рассылка.
Также может быть маршрутизируемой и немаршрутизируемой и плоскость данных. В маршрутизируемой плоскости данных есть основной участник. Передача данных происходит только между ним и остальными. Трафик при этом может быть и одно-, и двунаправленным. Пример маршрутизируемой плоскости данных – ATM.

В немаршрутизируемой плоскости данных любой член группы вправе передавать данные любому другому члену. Многоадресная рассылка в сетях

ATM маршрутизируется и в плоскости управления, и в плоскости данных, в то время как в сетях IP она не маршрутизируется ни в одной из плоскостей.

При перечислении записей протоколов информацию о поддержке многоадресной рассылки можно извлечь из поля dwServiceFlags структуры WSAPROTOCOL_INFO (установлен бит XP1_SUPPORT_MULTIPOINT). Установка бита XP1_MULTIPOINT_CONTROL_PLANE означает поддержку маршрутизируемой плоскости управления, а установка бита XP1_MULTIPOINT_DATA_PLANE – маршрутизируемой плоскости данных.
Многоадресная рассылка в сетях IP

Для многоадресной рассылки в сетях IP используются групповые адреса (multicast address). Они должны лежать в диапазоне 224.0.0.0 – 239.255.255.255. При этом часть этих адресов зарезервирована для специальных целей. Так, например, адрес 224.0.0.0 –базовый, 224.0.0.1 используется как групповой адрес всех систем данной подсети, 224.0.0.2 – как групповой адрес всех маршрутизаторов данной подсети, 224.0.1.24 – как групповой адрес WINS-сервера. IP требует, чтобы специальные адреса использовались только для многоадресного трафика.

Для управления клиентами многоадресной рассылки и их членством в группах был разработан специальный протокол IGMP (Internet Group Management Protocol). Он позволяет осуществлять действия, для которых нельзя воспользоваться средствами IP, например, присоединить рабочие станции из разных подсетей к одной многоадресной группе.

Протокол IGMP

Узлы многоадресной рассылки используют IGMP для сообщения маршрутизаторам, что компьютер маршрутизируемой подсети хочет присоединиться к определенной многоадресной группе. Необходимо, чтобы протокол IGMP поддерживался всеми маршрутизаторами на пути между узлами, обменивающимися информацией, в противном случае данные многоадресной рассылки будут ими просто отбрасываться.
Для того, чтобы присоединить приложение к многоадресной группе, необходимо отправить в адрес всех маршрутизаторов данной подсети (224.0.0.2) IGMP-команду join. При этом следует указать параметр время жизни (time-to-live, TTL) – сколько маршрутизаторов может пересекать сообщение на пути к точке назначения. Далее маршрутизаторы сами транслируют данную команду, уменьшая TTL на единицу.

Для сообщения о выходе из группы клиент отправляет маршрутизатору сообщение leave (поддерживается версией 2 протокола IGMP). Это позволяет избежать пересылки ненужной информации, как может быть при

использовании первой версии протокола. Версия 2 протокола IGMP поддерживается, начиная с Windows 9.

Листовые узлы IP

Поскольку любой участник многоадресной рассылки в сети IP является листовым (leaf) узлом, процесс присоединения к группе для всех одинаков. Он несколько различается при использовании Winsock первой или второй версии. При использовании Winsock 1 следует выполнить следующие шаги:
1. Посредством функции socket создать сокет семейства адресов AF_INET типа SOCK_DGRAM.
2. Связать сокет с локальным портом.
3. Вызвать функцию setsockopt с параметром IP_ADD_MEMBERSHIP и указанием адресной структуры для группы, к которой нужно присоединиться.

При использовании Winsock 2 на третьем шаге вместо функции setsockopt используется вызов WSAJoinLeaf для добавления своего адреса в группу.

Если приложение только отправляет данные, то присоединяться к группе не требуется – достаточно просто отправлять UDP-пакеты по групповому адресу. Многоадресная рассылка в сетях IP, как и обычный протокол UDP не требует логического соединения и так же ненадежна.

Многоадресная рассылка в сетях ATM

ATM также поддерживает многоадресную рассылку, причем предоставляет существенно большие возможности по сравнению с IP. ATM поддерживает маршрутизируемые плоскости управления и данных. При этом узел c_root фактически исполняет роль многоадресного сервера, который контролирует состав групп, а также маршруты передачи внутри группы.

4.4.Сетевой порядок байт

У разных процессоров процессоров различные порядки следования младших и старших байт. Так, например, у микропроцессоров Intel младшие байты располагаются по меньшим адресам, а у микропроцессоров Motorola 68000 - наоборот. Поскольку это вызывает проблемы при межсетевом взаимодействии, был введен специальный сетевой порядок байт, предписывающий старший байт передавать первым (не так, как у Intel!).
Для преобразований чисел из сетевого формата в формат локального хоста и наоборот предусмотрено четыре функции - первые две манипулируют короткими целыми (16-битными словами), а две последние - длинными (32-

битными двойными словами): u_short ntohs (u_short netshort); u_short htons (u_short hostshort); u_long ntohl (u_long netlong); u_long htonl (u_long hostlong); Отметим, что за буквой "n" скрывается сокращение "network", за "h" - "host" (подразумевается локальный), "s" и "l" соответственно короткое (short) и длинное (long) беззнаковые целые, а "to" и обозначает преобразование. Например, "htons" расшифровывается так: "Host Network (short)" т.е. преобразовать короткое целое из формата локального хоста в сетевой формат. Все значения, возвращенные socket-функциями, уже находятся в сетевом формате. Чаще всего к вызовам этих функций прибегают для преобразования номера порта согласно сетевому порядку. Например: dest_addr.sin_port = htons(110).

5. Примеры приложений.

Листинги1:
клиент с широковещанием.

```
#include "stdafx.h"
#include"winsock2.h"
 #include<iostream>
 #include<conio.h>
 using namespace std;
#define MYPORT 9009

 int main()
 { WSADATA wsaData;
 WSAStartup(MAKEWORD(2,2), &wsaData);
 SOCKET sock;
 sock = socket(AF_INET,SOCK_DGRAM,0);
 char broadcast = '1';

if(setsockopt(sock,SOL_SOCKET,SO_BROADCAST,&broadcast,sizeof(broadcas
t)) < 0)
 { cout<<"Error in setting Broadcast option";
 closesocket(sock);
 return 0; }
 struct sockaddr_in Recv_addr;
 struct sockaddr_in Sender_addr;
 int len = sizeof(struct sockaddr_in);
 char sendMSG[] ="Broadcast message from SLAVE TAG";
 char recvbuff[50] = "";
```

```
 int recvbufflen = 50;
 Recv_addr.sin_family = AF_INET;
 Recv_addr.sin_port = htons(MYPORT);
 Recv_addr.sin_addr.s_addr = INADDR_BROADCAST;
 sendto(sock,sendMSG,strlen(sendMSG)+1,0,(sockaddr
*)&Recv_addr,sizeof(Recv_addr));
 recvfrom(sock,recvbuff,recvbufflen,0,(sockaddr *)&Recv_addr,&len);
 cout<<"\n\n\tReceived message from Reader is => "<<recvbuff;
 cout<<"\n\n\tpress any key to CONT...";
 _getch();
 closesocket(sock);
 WSACleanup();
 }
```

Листинг 2

Сервер с широковещанием

```
#include "stdafx.h"
#include<winsock2.h>
#include<iostream>
#include<conio.h>
 using namespace std;
 #define MYPORT 9009

 int main()
 {
        WSADATA wsaData;
 WSAStartup(MAKEWORD(2,2), &wsaData);
 SOCKET sock; sock = socket(AF_INET,SOCK_DGRAM,0);
 char broadcast = '1';

if(setsockopt(sock,SOL_SOCKET,SO_BROADCAST,&broadcast,sizeof(broadcas
t)) < 0)
 { cout<<"Error in setting Broadcast option";
 closesocket(sock);
 return 0; }
 struct sockaddr_in Recv_addr;
 struct sockaddr_in Sender_addr;
 int len = sizeof(struct sockaddr_in);
 char recvbuff[50];
 int recvbufflen = 50;
 char sendMSG[]= "Broadcast message from READER";
```

```
 Recv_addr.sin_family = AF_INET;
 Recv_addr.sin_port = htons(MYPORT);
 Recv_addr.sin_addr.s_addr = INADDR_ANY;
 if (bind(sock,(sockaddr*)&Recv_addr, sizeof (Recv_addr)) < 0)
 { cout<<"Error in BINDING"<<WSAGetLastError();
 _getch();
 closesocket(sock);
 return 0; }
 recvfrom(sock,recvbuff,recvbufflen,0,(sockaddr *)&Sender_addr,&len);
 cout<<"\n\n\tReceived Message is : "<<recvbuff; cout<<"\n\n\tPress Any to send
message"; _getch();
 if(sendto(sock,sendMSG,strlen(sendMSG)+1,0,(sockaddr
*)&Sender_addr,sizeof(Sender_addr)) < 0)
 { cout<<"Error in Sending."<<WSAGetLastError(); cout<<"\n\n\t\t Press any key
to continue...."; _getch();
 closesocket(sock);
 return 0; }
 else cout<<"\n\n\n\tREADER sends the broadcast message Successfully";
 cout<<"\n\n\tpress any key to CONT...";
 _getch();
 closesocket(sock);
 WSACleanup();
 return 0;
 }
```

Листинг 3
Многоадресная рассылка.

```
#include "stdafx.h"
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include <stdlib.h>

#define MCASTADDR "234.5.6.7"
#define MCASTPORT 25000
#define BUFSIZE 1024
#define DEFAULT_COUNT 500
```

```
BOOL bSender = FALSE;
BOOL bLoopBack = FALSE;

DWORD dwInterface;
DWORD dwMulticastGroup;
DWORD dwCount;

short iPort;

 // Вывод информации об использовании и выход
void usage(char *progname) {
        printf("usage: %s -s -m:str -p:int -i:str -l -n:int\n",progname);
        printf(" -s      Act as server (send data); otherwise\n");
        printf("         receive data.\n");
        printf(" -m:str  Dotted decimal multicast IP address to join\n");
        printf("         The default group is: %s\n", MCASTADDR);
        printf(" -p:int  Port number to use\n");
        printf("         The default port is: %d\n", MCASTPORT);
        printf(" -i:str  Local interface to bind to; by default\n");
        printf("         use INADDR_ANY\n");
        printf(" -l      Disable loopback\n");
        printf(" -n:int  Number of messages to send/receive\n");
        ExitProcess(-1);
}

/*Анализ параметров командной строки и установка
 некоторых глобальных флагов в зависимости от значений*/
void ValidateArgs(int argc,char **argv) {
        int i;
        dwInterface = INADDR_ANY;
        dwMulticastGroup = inet_addr(MCASTADDR);
        iPort = MCASTPORT;
        dwCount = DEFAULT_COUNT;

        for(i=1; i<argc; i++) {
                if((argv[i][0] == '-') || (argv[i][0] == '/')) {
                        switch (tolower(argv[i][1])) {

                        case 's':
                                bSender = TRUE;
                                break;
```

```
                        case 'm':
                                if(strlen(argv[i]) > 3)
                                        dwMulticastGroup = inet_addr(&argv[i][3]);
                                break;

                        case 'i':
                                if(strlen(argv[i]) > 3)
                                        dwInterface = inet_addr(&argv[i][3]);
                                break;

                        case 'p':
                                if(strlen(argv[i]) > 3)
                                        iPort =  atoi(&argv[i][3]);
                                break;

                        case 'l':
                                bLoopBack = TRUE;
                                break;

                        case 'n':
                                dwCount = atoi(&argv[i][3]);
                                break;

                        default:
                                usage(argv[0]);
                                break;
                        }
                }
        }
        return;
}

int main(int argc,char **argv) {
        WSADATA wsd;
        struct sockaddr_in local,remote,from;
        SOCKET sock,sockM;
        TCHAR recvbuf[BUFSIZE],sendbuf[BUFSIZE];
        int len = sizeof(struct sockaddr_in);
        int optval;
        int ret;
```

```
        DWORD i=0;

        ValidateArgs(argc,argv);

        if(WSAStartup(MAKEWORD(2,2), &wsd)!=0) {
                printf("WSAStartup() failed\n");
                return -1;
        }

        if((sock = WSASocket(AF_INET, SOCK_DGRAM, 0, NULL, 0,
                WSA_FLAG_MULTIPOINT_C_LEAF
                | WSA_FLAG_MULTIPOINT_D_LEAF
                | WSA_FLAG_OVERLAPPED)) == INVALID_SOCKET) {
                printf("socket failed with^ %d\n", WSAGetLastError());
                WSACleanup();
                return -1;
        }

        local.sin_family = AF_INET;
        local.sin_port = htons(iPort);
        local.sin_addr.s_addr = dwInterface;

        if(bind(sock, (struct sockaddr *)&local, sizeof(local)) ==SOCKET_ERROR)
{
                printf("bind failed with: %d\n", WSAGetLastError());
                closesocket(sock);
                WSACleanup();
                return -1;
        }

        remote.sin_family = AF_INET;
        remote.sin_port = htons(iPort);
        remote.sin_addr.s_addr = dwMulticastGroup;

        optval = 8;

        if(setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL,
                (char *)&optval, sizeof(int)) == SOCKET_ERROR) {
                printf("setsockopt(IP_MULTICASTT_TTL) failed: %d\n",
                        WSAGetLastError());
                closesocket(sock);
```

```
            WSACleanup();
            return -1;
        }

        if(bLoopBack) {
            optval = 0;

            if(setsockopt(sock, IPPROTO_IP, IP_MULTICAST_LOOP,
                (char *)&optval, sizeof(optval)) == SOCKET_ERROR) {
                printf("setsockopt(IP_MULTICAST_LOOP) failed: %d\n",
WSAGetLastError());
                closesocket(sock);
                WSACleanup();
                return -1;
            }
        }

        if((sockM = WSAJoinLeaf(sock, (SOCKADDR *)&remote, sizeof(remote),
            NULL, NULL, NULL, NULL,JL_BOTH)) == INVALID_SOCKET) {
            printf("WSAJoinLeaf() failed: %d\n", WSAGetLastError());
            closesocket(sock);
            WSACleanup();
            return -1;
        }

        if(!bSender) {
            for(i=0;i<dwCount; i++) {
                if ((ret = recvfrom(sock, recvbuf, BUFSIZE, 0,
                    (struct sockaddr *)&from, &len)) == SOCKET_ERROR)
{
                    printf("recvfrom failed with: %d\n", WSAGetLastError());
                    closesocket(sockM);
                    closesocket(sock);
                    WSACleanup();
                    return -1;
                }

                recvbuf[ret] = 0;
                printf("RECV: '%s' from <%s>\n", recvbuf,
inet_ntoa(from.sin_addr));
            }
```

```
        }
        else {
                for(i=0;i=dwCount;i++) {
                        sprintf(sendbuf, "server 1:This is a test: %d", i);

                        if(sendto(sock, (char *)sendbuf, strlen(sendbuf), 0,
                                (struct sockaddr *)&remote, sizeof(remote)) ==
SOCKET_ERROR) {
                                printf("sendto failed with: %d\n",
                                        WSAGetLastError());
                                closesocket(sockM);
                                closesocket(sock);
                                WSACleanup();
                                return -1;
                        }
                        Sleep(500);
                }
        }
        closesocket(sock);
        WSACleanup();
        return -1;
        TCP/IP Крупным планом - Широковещательная и групповая адресация
```

Листинг 4
Информация о протоколах

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <objbase.h>
#include <stdio.h>

// Link with ws2_32.lib and ole32.lib
#pragma comment (lib, "Ws2_32.lib")
#pragma comment (lib, "ole32.lib")

#define MALLOC(x) HeapAlloc(GetProcessHeap(), 0, (x))
#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))
// Note: could also use malloc() and free()

int wmain()
{
```

```
//------------------------------------
// Declare and initialize variables
WSADATA wsaData;
int iResult = 0;

int iError = 0;
INT iNuminfo = 0;

int i;

// Allocate a 16K buffer to retrieve all the protocol providers
DWORD dwBufferLen = 16384;

LPWSAPROTOCOL_INFO lpProtocolInfo = NULL;

// variables needed for converting provider GUID to a string
int iRet = 0;
WCHAR GuidString[40] = { 0 };

// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (iResult != 0) {
    wprintf(L"WSAStartup failed: %d\n", iResult);
    return 1;
}

lpProtocolInfo = (LPWSAPROTOCOL_INFO) MALLOC(dwBufferLen);
if (lpProtocolInfo == NULL) {
    wprintf(L"Memory allocation for providers buffer failed\n");
    WSACleanup();
    return 1;
}

iNuminfo = WSAEnumProtocols(NULL, lpProtocolInfo, &dwBufferLen);
if (iNuminfo == SOCKET_ERROR) {
    iError = WSAGetLastError();
    if (iError != WSAENOBUFS) {
        wprintf(L"WSAEnumProtocols failed with error: %d\n", iError);
        if (lpProtocolInfo) {
            FREE(lpProtocolInfo);
```

```
            lpProtocolInfo = NULL;
        }
        WSACleanup();
        return 1;
    } else {
        wprintf(L"WSAEnumProtocols failed with error: WSAENOBUFS (%d)\n",
                iError);
        wprintf(L"  Increasing buffer size to %d\n\n", dwBufferLen);
        if (lpProtocolInfo) {
            FREE(lpProtocolInfo);
            lpProtocolInfo = NULL;
        }
        lpProtocolInfo = (LPWSAPROTOCOL_INFO) MALLOC(dwBufferLen);
        if (lpProtocolInfo == NULL) {
            wprintf(L"Memory allocation increase for buffer failed\n");
            WSACleanup();
            return 1;
        }
        iNuminfo = WSAEnumProtocols(NULL, lpProtocolInfo, &dwBufferLen);
        if (iNuminfo == SOCKET_ERROR) {
            iError = WSAGetLastError();
            wprintf(L"WSAEnumProtocols failed with error: %d\n", iError);
            if (lpProtocolInfo) {
                FREE(lpProtocolInfo);
                lpProtocolInfo = NULL;
            }
            WSACleanup();
            return 1;
        }

    }
}

wprintf(L"WSAEnumProtocols succeeded with protocol count = %d\n\n",
        iNuminfo);
for (i = 0; i < iNuminfo; i++) {
    wprintf(L"Winsock Catalog Provider Entry #%d\n", i);
    wprintf
        (L"----------------------------------------------------\n");
    wprintf(L"Entry type:\t\t\t ");
    if (lpProtocolInfo[i].ProtocolChain.ChainLen == 1)
```

```
   wprintf(L"Base Service Provider\n");
else
   wprintf(L"Layered Chain Entry\n");

wprintf(L"Protocol:\t\t\t %ws\n", lpProtocolInfo[i].szProtocol);

iRet =
   StringFromGUID2(lpProtocolInfo[i].ProviderId,
               (LPOLESTR) & GuidString, 39);
if (iRet == 0)
   wprintf(L"StringFromGUID2 failed\n");
else
   wprintf(L"Provider ID:\t\t\t %ws\n", GuidString);

wprintf(L"Catalog Entry ID:\t\t %u\n",
      lpProtocolInfo[i].dwCatalogEntryId);

wprintf(L"Version:\t\t\t %d\n", lpProtocolInfo[i].iVersion);

wprintf(L"Address Family:\t\t\t %d\n",
      lpProtocolInfo[i].iAddressFamily);
wprintf(L"Max Socket Address Length:\t %d\n",
      lpProtocolInfo[i].iMaxSockAddr);
wprintf(L"Min Socket Address Length:\t %d\n",
      lpProtocolInfo[i].iMinSockAddr);

wprintf(L"Socket Type:\t\t\t %d\n", lpProtocolInfo[i].iSocketType);
wprintf(L"Socket Protocol:\t\t %d\n", lpProtocolInfo[i].iProtocol);
wprintf(L"Socket Protocol Max Offset:\t %d\n",
      lpProtocolInfo[i].iProtocolMaxOffset);

wprintf(L"Network Byte Order:\t\t %d\n",
      lpProtocolInfo[i].iNetworkByteOrder);
wprintf(L"Security Scheme:\t\t %d\n",
      lpProtocolInfo[i].iSecurityScheme);
wprintf(L"Max Message Size:\t\t %u\n", lpProtocolInfo[i].dwMessageSize);

wprintf(L"ServiceFlags1:\t\t\t 0x%x\n",
      lpProtocolInfo[i].dwServiceFlags1);
wprintf(L"ServiceFlags2:\t\t\t 0x%x\n",
      lpProtocolInfo[i].dwServiceFlags2);
```

```
        wprintf(L"ServiceFlags3:\t\t\t 0x%x\n",
              lpProtocolInfo[i].dwServiceFlags3);
        wprintf(L"ServiceFlags4:\t\t\t 0x%x\n",
              lpProtocolInfo[i].dwServiceFlags4);
        wprintf(L"ProviderFlags:\t\t\t 0x%x\n",
              lpProtocolInfo[i].dwProviderFlags);

        wprintf(L"Protocol Chain length:\t\t %d\n",
              lpProtocolInfo[i].ProtocolChain.ChainLen);

        wprintf(L"\n");
    }

    if (lpProtocolInfo) {
        FREE(lpProtocolInfo);
        lpProtocolInfo = NULL;
    }
    WSACleanup();

    return 0;
}
```

6. Список вопросов к защите лабораторной работы

1. Особенности механизма широковещания.
2. Особенности многоадресной рассылки в сетях с различными протоколами.
3. Способ организации членства в группах.
4. Маршрутизируемые и немаршрутизируемые рассылки данных.
5. Структура приложения - клиент широковещательной рассылки.
6. Структура приложения - сервер широковещательной рассылки.
7. Структура приложения многоадресной рассылки.

Литература.

1.В.Г.Олифер, Н.А. Олифер. Сетевые операционные системы. - Спб: Питер, 2003.

2.Сетевые средства Microsoft Windows NT Server 4.0 / Пер. с англ. - Спб: Издательская группа BHV,1999.

3.Джонс Э., Оланд Дж. Программирование в сетях Microsoft Windows / Пер. с англ. -Спб:: Издательско-торговый дом «Русская редакция», 2002.

4.Рихтер Дж., Кларк Дж. Д. Программирование серверных приложений для Microsoft Windows 2000/ Пер. с англ. - Спб: : Издательско-торговый дом «Русская редакция», 2001.

5.Джонсон М. Хард. Системное программирование в среде Win 32. / Пер. с англ. - М: : Издательский дом «Вильямс», 2001.