**7. РАЗРЕШЕНИЕ АДРЕСОВ В IP-СЕТЯХ**

**7.1. Определение MAC-адреса удаленного хоста по IP-адресу**

***7.1.1. Физический адрес***

*Физический*,или*локальный адрес узла* определяется технологией, с помощью которой построена сеть, в которую входит узел. Для узлов, входящих в локальные сети, это МАС-адрес сетевого адаптера или порта маршрутизатора.

В качестве стандартного выбран 48-битный формат адреса, что соответствует примерно 280 триллионам различных адресов. Понятно, что столько сетевых адаптеров никогда не будет выпущено.

С тем, чтобы распределить возможные диапазоны адресов между многочисленными изготовителями сетевых адаптеров, была предложена следующая структура адреса (рис. 7.1).

I/G

U/L

OUI (уникальный идентификатор)

OUA (уникальный адрес)

1 бит

1 бит

22 бита

24 бита

UAA (46 бит)

Рис. 7.1.  Структура 48-битного стандартного MAC-адреса

Младшие 24 разряда кода адреса называются OUA (Organizationally Unique Address) – *уникальный адрес*. Именно их присваивает каждый из зарегистрированных производителей сетевых адаптеров. Всего возможно свыше 16 миллионов комбинаций, то есть каждый изготовитель может выпустить 16 миллионов сетевых адаптеров.

Следующие 22 разряда кода называются OUI (Organizationally Unique Identifier) – *уникальный идентификатор*. IEEE присваивает один или несколько OUI каждому производителю сетевых адаптеров. Это позволяет исключить совпадения адресов адаптеров от разных производителей. Всего возможно свыше 4 миллионов разных OUI, это означает, что теоретически может быть зарегистрировано 4 миллиона производителей. Вместе OUA и OUI называются UAA (Universally Administered Address) – универсально управляемый адрес или IEEE-адрес.

Два старших разряда адреса управляющие, они определяют тип адреса, способ интерпретации остальных 46 разрядов. Старший бит I/G (Individual/Group) указывает на тип адреса. Если он установлен в 0, то индивидуальный, если в 1, то групповой (многопунктовый или функциональный). Пакеты с групповым адресом получат все имеющие этот групповой адрес сетевые адаптеры. Причем групповой адрес определяется 46-ю младшими разрядами. Второй управляющий бит U/L (Universal/Local) называется флажком универсального/местного управления и определяет, как был присвоен адрес данному сетевому адаптеру. Обычно он установлен в 0. Установка бита U/L в 1 означает, что адрес задан не производителем сетевого адаптера, а организацией, использующей данную сеть. Это случается довольно редко.

Для широковещательной передачи (то есть передачи всем абонентам сети одновременно) применяется специально выделенный *сетевой адрес*, все 48 битов которого установлены в единицу. Его принимают все абоненты сети независимо от их индивидуальных и групповых адресов.

***7.1.2. Разработка программы определения MAC-адреса компьютера по IP-адресу и наоборот***

**Windows Sockets API**

В ходе выполнения поставленной задачи необходимо использовать интерфейс Windows Sockets API (WAS).

Winsock (Windows socket) – это интерфейс программирования приложений (API) созданный для реализации сетевых приложений на основе протокола TCP/IP. Интерфейс реализуется библиотекой Wsock32.dll, экспортирующей функции работы с сокетами.

Сокет – это конечная точка в сетевом взаимодействии. Главная задача сокета – инкапсулировать реализацию установления соединения, обмена пакетами, разрыв и оповещение о происходящих ошибках в процессе сетевого взаимодействия в различных протоколах, предоставив разработчику простой и удобный интерфейс. Сокет характеризуется различными параметрами (семейство адресов, тип протоколов для передачи данных и др.), среди которых сетевой адрес и порт (при использовании транспортного протокола, требующего установления соединения).

Для подключения библиотек, содержащих необходимые структуры и функции, следует указать следующие директивы:

|  |
| --- |
| #include <Winsock2.h>  #include <Iphlpapi.h>  #pragma comment(lib, "WS2\_32.lib")  #pragma comment(lib, "IPHlpApi.Lib") |

Включение заголовочного файла <Winsock2.h> необходимо для использования функций Windows Socket версии 2. Единственное различие с заголовочным файлом <Winsock.h> то, что он содержит больше кодов ошибок новых API-функций. Заголовочный файл <Iphlpapi.h> нужен для использования API IP helper. Он также подключает другие заголовочные файлы, содержащие нужные структуры и перечисления.

Приложение, использующее Winsock, должно быть скомпоновано с библиотекой экспорта "WS2\_32.lib". Для сборки также требуется файл "IPHlpApi.Lib".

**Инициализация Winsock**

Чтобы начать использовать интерфейс сокетов, необходимо инициализировать библиотеку WS2\_32.dll с помощью вызова функции WSAStartup, передав ей в старшем байте номер требуемой версии, а в младшем – подверсии (первый аргумент):

|  |
| --- |
| WSADATA wsaData;  WSAStartup(0x0202, &wsaData); |

Таким образом, первый параметр функции 0x0202 запрашивает версию 2.2 Winsock (можно использовать макрос MAKEWORD(2, 2)).

В большинстве случаев при написании новых приложений целесообразно загружать последнюю доступную версию библиотеки Winsock. Если используется, например, версия 3, приложение, использующее версию 2.2, должно выполняться корректно. При запросе более поздней версии Winsock, не поддерживаемой вашей платформой, WSAStartup вернет ошибку, а в поле wHighVersion структуры wsaData появится номер последней версии библиотеки, поддерживаемой данной системой.

Аргумент wsaData должен указывать на структуру WSADATA, в которую при успешной инициализации будет занесена информация о производителе библиотеки. Индивидуальные поля структуры WSADATA содержат: версию Winsock; высшую версию Winsock, поддерживаемую загруженной библиотекой; текстовое описание загруженной библиотеки; текстовую строку с соответствующей информацией о состоянии или конфигурации; максимальное количество сокетов; максимальный размер дейтаграммы UDP; информацию об изготовителе.

Если инициализация не удалась, то функция возвращает ненулевое значение. Например:

|  |
| --- |
| // Инициализация Winsock  WSADATA WsaData;  if(WSAStartup(0x0202,&WsaData)!=NULL)  cout<<"WSA: Ошибка !\n"; |

С этого момента можно начинать использовать весь спектр функций, предоставляемых WSA.

**Создание сокета**

Следующий шаг – создаем сокет. Сокет представлен отдельным типом – SOCKET и создается одной из двух функций:

|  |
| --- |
| SOCKET WSASocket  (  \_\_in int af,  // определяет семейство адресов протокола  \_\_in int type,  //тип сокета для данного протокола  \_\_in int protocol,  //указывает конкретный транспорт  \_\_in LPWSAPROTOCOL\_INFO lpProtocolInfo,  \_\_in GROUP g,  //параметр группы всегда равен 0  \_\_in DWORD dwFlags  //флаги  ); |

или

|  |
| --- |
| SOCKET WSAAPI socket  (  \_\_in int af,  // определяет семейство адресов протокола  \_\_in int type,  //тип сокета для данного протокола  \_\_in int protocol  //указывает конкретный транспорт  ); |

Первый аргумент указывает на семейство используемых протоколов, константа AF\_INET используется, чтобы сослаться на протокол IP (IPv4) (AF\_INET6 для IPv6) и создать UDP- или ТСР-сокет.

Второй аргумент – тип создаваемого сокета. Принимает значения: SOCK\_STREAM (потоковый), или SOCK\_DGRAM (дейтаграммный), или SOCK\_RAW (сырой) и др (подробный перечень значений приведен в MSDN).

Последний аргумент задает тип создаваемого сокета. Нулевое значение соответствует выбору по умолчанию (система выбирает поставщика транспорта исходя из других двух параметров  af и type): ТСР – для потоковых, UDP – дейтаграммных.

Если функция завершилась успешно, то она возвращает дескриптор сокета. Например пользователь вводит IP адрес, проверяем корректность структуры и диапазона заданного адреса (CheckAddr) и открываем сокет:

|  |
| --- |
| char \*ip\_,\*ip\_end\_;  DWORD ip,ip\_end;  bool flag=true;  ip\_=new char[16];  SOCKET udp\_s;  SOCKADDR\_IN udp\_sin;  do  {  if(!flag) cout<<"Неверный IP"<<endl;  cout<<"Введите IP: ";  cin>>ip\_;  }while(!(flag=CheckAddr(ip\_)));  udp\_s=socket(AF\_INET,SOCK\_DGRAM,IPPROTO\_UDP);  if (udp\_s!=SOCKET\_ERROR)  {  //Сокет успешно открыт  }  else  {  cout<<"Ошибка открытия socket\n";  } |

**Установка соединения**

Следующий шаг – установка соединения.

*Адресация*

Сетевым адаптерам назначается IP-адрес, состоящий из 32 бит, официально называемый IP-адресом версии 4 (IPv4). Для взаимодействия с сервером по TCP или UDP клиент должен указать IP-адрес сервера и номер порта. Чтобы прослушивать входящие запросы клиента, сервер также должен указать свой IP-адрес и номер порта.

В Winsock IP-адрес и порт задают в структуре sockaddr\_in:

|  |
| --- |
| struct sockaddr\_in  {  short sin\_family;  //семейство адресов  u\_short sin\_port;  //коммуникационный порт TCP или UDP,  //который будет использован  //для идентификации службы сервера  struct in\_addr sin\_addr;  //адресс  char sin\_zero[8];  }; |

Поле sin\_family должно быть равно AF\_INET: этим Winsock сообщает об использовании семейства адресов IP.

*Порты*

При использовании TCP и UDP приложение решает, через какой порт установить связь. Существуют стандартные номера портов, зарезервированные для служб сервера, поддерживающих протоколы более высокого уровня, чем TCP. Например, порт 21 зарезервирован для FTP, 80 – для HTTP. Стандартные службы обычно используют порты 1-1023. Для того чтобы узнать номера портов, используемых стандартными службами, можно вызвать функцию getservbyname или WSAAsyncGetServByName. Эти функции просто извлекают статическую информацию из файла с именем services.

Во избежание накладок с портами, уже занятыми системой или другим приложением, ваша программа должна выбирать зарегистрированные порты в диапазоне 1024 –  49151. Порты 49152 – 65535 также можно задействовать свободно – с ними не связаны никакие стандартные службы.

Поле sin\_addr хранит IP-адрес в 4-байтном виде с типом unsigned long int (может представлять и локальный, и удаленный IP-адрес). IP-адреса задают в десятично-точечной нотации типа a.b.c.d. Здесь каждая буква представляет число для каждого байта и задается слева направо (все четыре байта с типом unsigned long int). И наконец, поле sin\_zero играет роль простого заполнителя, чтобы структура sockaddr\_in по размеру равнялась исходной структуре sockaddr.

В разрабатываемой программе создадим объект udp\_sin, содержащий структуру типа sockaddr\_in, и выполним инициализацию:

|  |
| --- |
| ..................................................  if(udp\_s!=SOCKET\_ERROR)  {  ip=inet\_addr(ip\_);  udp\_sin.sin\_family = AF\_INET; //семейство  udp\_sin.sin\_port = htons(5234); //порт  udp\_sin.sin\_addr.S\_un.S\_addr = ip; //IP адресс  } |

При инициализации определяется семейство адреса, IP-адрес и порт для соединения. Здесь \_ip должен быть предвартельно сформировон на основе введенных пользователем данных:

*Преобразование адреса*

Полезная вспомогательная функция inet\_addr преобразует IP-адрес из точечной нотации в 32-битное длинное целое без знака и имеет формат:

|  |
| --- |
| unsigned long inet\_addr( const char FAR \*cp ); |

Поле cp является строкой, заканчивающейся нулевым символом. Функция в качестве результата возвращает IP-адрес, представленный 32-битным числом с сетевым порядком следования байт (network-byte order). Функция является устаревшей, для подавления предупреждений установите:

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

Для конвертации IP адреса желательно использовать функции:

#include <arpa/inet.h>

const char \*inet\_ntop (int af, const void \*src,

char \*dst, socklen\_t size);

int inet\_pton (int af, const char \*src, void \*dst);

*Порядок байт*

Разные процессоры в зависимости от конструкции представляют числа в одном из двух порядков байт big-endian или little-endian. Номер порта и IP-адрес хранятся в памяти компьютера как многобайтные числа и представляются в системном порядке (host-byte-order). Когда IP-адрес или номер порта задаются по сети, стандарты Интернета требуют, чтобы многобайтные значения представлялись от старшего байта к младшему (в порядке big-endian), что обычно называется сетевым порядком (network-byte order).

Есть целый ряд функций для преобразования многобайтных чисел из системного порядка в сетевой, и обратно. Например следующая API-функция htons преобразует числа из системного порядка в сетевой:

|  |
| --- |
| u\_short WSAAPI htons  (  \_\_in u\_short hostshort// двухбайтное число  // с системным порядком  ); |

Функция htons возвращает число как двухбайтное значение с сетевым порядком.

**Передача данных**

Следующий шаг при решени задачи – передача данных. По сути, в сетевом программировании самое главное – уметь отправлять и принимать данные. Для пересылки данных по сокету используются функции send, sendto и WSASend. Аналогично, для приема данных существуют функции recv, recvfrom и WSARecv. Все буферы, используемые при отправке и приеме данных, состоят из элементов типа char.

Будем использовать функцию sendto, она имеет следующий формат:

|  |
| --- |
| int sendto  (  \_\_in SOCKET s, //дескриптор сокета  \_\_in const char \*buf,  //указатель на буффер,  // содержащий данные для пересылки  \_\_in int len,  //длинна в байтах передавемого буффера  \_\_in int flags,  //флаги, которые определяют  //способ вызова  \_\_in const struct sockaddr \*to,  //указатель на структуру SOCKADDR  \_\_in int tolen  //размер структуры SOCKADDR  ); |

В приложении отсылаем сообщение. Для этого мы вызовем функцию sendto.

|  |
| --- |
| if  (sendto(udp\_s,"TEST",5,NULL,(SOCKADDR\*)&udp\_sin,  sizeof(udp\_sin))>0)  {    }  else  {  cout << "Ошибка передачи! \n " << WSAGetLastError();  } |

Здесь udp\_s – идентификатор определенного ранее сокета. Если ошибок не было sendto, возвратится общее число переданных байт, которое может быть меньше, чем указано в третьем аргументе функции. В противном случае будет возвращено значение SOCKET\_ERROR, а код ошибки может быть извлечен вызовом функции WSAGetLastError или функцией GetLastError .

Для ТСР-клиента (UDP-клинета) можно использовать функцию connect (версия Winsock 1) или WSAConnect (версия Winsock 2). У нее первый элемент – это дескриптор сокета для установления соединения, второй – структура адреса сокета sockaddr, содержащая адрес (IP), порт, к которому подключаются, и последний аргумент, сообщающий о размере sockaddr. Если по каким-то причинам установить соединение не удается, то функция возвращает ненулевое значение:

|  |
| --- |
| if (connect (udp\_sock, (sockaddr\*)& udp\_sin,  sizeof (udp\_sin)) )  {  cout << "Ошибка соединения! \n "  << WSAGetLastError();  return -1;  } |

Для получения MAC-адреса в дальнешем будем использовать Internet Protocol Helper (IP helper) API. Он нозволяет извлекать и модифицировать сетевую конфигурацию для локального компьютера.

**Получение таблицы ARP**

Функция GetIpNetTable (или GetIpNetTable2) извлекает IPv4 таблицу маппирования физических адресов. Прототип функции:

|  |
| --- |
| DWORD GetIpNetTable  (  \_\_out PMIB\_IPNETTABLE pIpNetTable,  //указатель на структуру  \_\_inout PULONG pdwSize,  //размер структуры  \_\_in BOOL bOrder  //порядок сортировки по IP  ); |

В случае успеха функция возвращает NO\_ERROR или ERROR\_NO\_DATA, если функция выполнилась успешно, но данных нет. Таблица возвращается через первый аргумент функции в виде структуры MIB\_IPNETTABLE, которая содержит записи Address Resolution Protocol (ARP) для IPv4 адресов:

|  |
| --- |
| typedef struct \_MIB\_IPNETTABLE  {  DWORD dwNumEntries;  //количество записей в таблице  MIB\_IPNETROW table[ANY\_SIZE];  //указатель на массив MIB\_IPNETROW структур  } MIB\_IPNETTABLE, \*PMIB\_IPNETTABLE; |

В свою очередь вложенная структура MIB\_IPNETROW содержит:

|  |
| --- |
| typedef struct \_MIB\_IPNETROW  {  DWORD dwIndex;  //индекс адаптера  DWORD dwPhysAddrLen;  //длина физического адреса в байтах  BYTE bPhysAddr[MAXLEN\_PHYSADDR];  //физический адрес  DWORD dwAddr;  //IPv4 адрес  DWORD dwType;  //тип ARP-записи  } MIB\_IPNETROW, \*PMIB\_IPNETROW; |

Тогда, для получения МАС-адреса вызываем функцию GetIpNetTable:

|  |
| --- |
| MIB\_IPNETTABLE \* pIpNetTable =  (MIB\_IPNETTABLE \*) new char[0xFFFF];  ULONG cbIpNetTable = 0xFFFF;  if(NO\_ERROR ==  GetIpNetTable(pIpNetTable,&cbIpNetTable,true))  {  for(DWORD i =0;i<pIpNetTable->dwNumEntries;i++)  {  //если адресс в поле dwAddr совпадает с искомым ip  //и dwType огтличен от 2.  //значение 2 соответсвует недоступной или  //незаконченной ARP записи  if(pIpNetTable->table[i].dwAddr == ip  && pIpNetTable->table[i].dwType!=2)  {  //извлекаем и выводим MAC  printf("%s : %X-%X-%X-%X-%X-%X\n",ip\_,  pIpNetTable->table[i].bPhysAddr[0],  pIpNetTable->table[i].bPhysAddr[1],  pIpNetTable->table[i].bPhysAddr[2],  pIpNetTable->table[i].bPhysAddr[3],  pIpNetTable->table[i].bPhysAddr[4],  pIpNetTable->table[i].bPhysAddr[5]); //освобожаем таблицу  delete[] pIpNetTable;  }  }  } |

**Освобождение Winsock**

По завершении работы с библиотекой Winsock вызовите функцию для выгрузки библиотеки и освобождения ресурсов. Функция WSACleanup заканчивает использование Winsock DLL (Ws2\_32.dll) (при удачном вызове возвращает ноль):

|  |
| --- |
| if (WSACleanup()== SOCKET\_ERROR)  cout<< "ERROR Closesocket "  <<WSAGetLastError ( ); |

Для каждого вызова WSAStartup необходимо согласованно вызывать WSACleanup, так как каждый стартовый вызов увеличивает значение эталонного счетчика ссылок на загруженные Winsock DLL. Чтобы уменьшить значение счетчика, требуется равное количество вызовов WSACleanup.

Обратите внимание: Winsock 2 полностью совместим со всеми вызовами функций Winsock 1.1. Поэтому приложение, написанное для Winsock 1.1, будет работать и с библиотекой Winsock 2 – функции Winsock 1.1 сопоставляются с их эквивалентами в Winsock 2.

Используя описанные в разделе функции и структуры, можно выполнить обратное определение: по заданному MAC-определить IP-адрес.

На рис. 7.2 приведен результат работы программы, выводящий МАС-адрес для заданного IP-адреса, на основе описанного подхода:

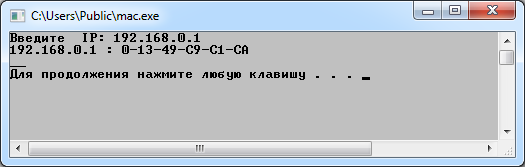


Рис. 7.2. Результат работы программы

**SendARP**

Для определения MAC-адреса можно использовать еще одну полезную функцию SendARP, которая посылает Address Resolution Protocol (ARP) запрос. На основе полученных данных можно получить MAC адресата по заданному IPv4 адресу:

|  |
| --- |
| DWORD SendARP  (  \_\_in IPAddr DestIP,  //IPv4 адрес адресата  \_\_in IPAddr SrcIP,  //IPv4 адрес источника  \_\_out PULONG pMacAddr,  //указатель на массив физического адреса  \_\_inout PULONG PhyAddrLen  //максимальная длина буффера массива физ. адреса  ); |

В случае успеха функция возвращает значение NO\_ERROR. Рассмотрим другой способ получения MAC:

|  |
| --- |
| #include <iostream>  using namespace std;  #include <winsock2.h>  #include <iphlpapi.h>  #include <stdio.h>  #include <conio.h>  #include <Ws2tcpip.h>  #pragma comment(lib , "iphlpapi.lib")  #pragma comment(lib , "ws2\_32.lib")  void GetMacAddress(unsigned char \*, struct in\_addr);  int main()  {  setlocale(LC\_ALL, "RUS");  unsigned char mac[6];  struct in\_addr srcip = { 0 };  struct sockaddr\_in sa;  char ip\_address[32];  WSADATA firstsock;  if (WSAStartup(MAKEWORD(2, 2), &firstsock) != 0)  {  cout<<"Ошибка инициализации winsock";  cout<< WSAGetLastError();  return -1;  }  cout<<"Введите IP : ";  cin>> ip\_address;  //преобразование IP адреса другим способом  //srcip.s\_addr = inet\_addr(ip\_address);    inet\_pton(AF\_INET, ip\_address, &(sa.sin\_addr));    //Получение MAC по IP  GetMacAddress(mac, sa.sin\_addr);  //GetMacAddress(mac, srcip);  printf("MAC адрес : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X",  mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);  printf("\n");  getch();  return 0;  }  void GetMacAddress(unsigned char \*mac, struct in\_addr destip)  {  DWORD ret;  IPAddr srcip;  ULONG MacAddr[2];  ULONG PhyAddrLen = 6;  int i;  srcip = 0;  //Послать ARP пакет  ret = SendARP((IPAddr)destip.S\_un.S\_addr,  srcip, MacAddr, &PhyAddrLen);  //Преобразовать адрес  if (PhyAddrLen)  {  BYTE \*bMacAddr = (BYTE \*)& MacAddr;  for (i = 0; i < (int)PhyAddrLen; i++)  {  mac[i] = (char)bMacAddr[i];  }  }  } |

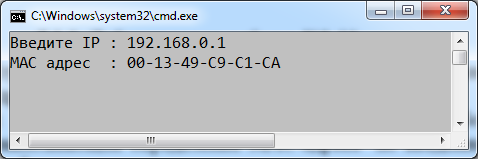
На рис. 7.3 приведен результат работы программы, выводящий МАС-адрес для заданного IP-адреса, на основе описанного подхода:

Рис. 7.3. Результат работы программы

***7.1.3. Лабораторная работа №9-10***

**Цель:** Изучение методов определения MAC-адреса удаленного хоста по IP-адресу и наоборот.

**Задание:** Лабораторная работа заключается в разработке программы, выполняющей определение MAC-адреса по введенному IP-адресу и наоборот.

**Дополнительное задание**: поиск MAC-адреса должен осуществляться для диапазона IP-адресов различных классов.