Белгородский государственный технологический университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №3 по теме: «Программирование протокола IP с использованием библиотеки Winsock»

**Выполнил:**  
студент группы ПВ-31  
Адаменко И. И.

**Проверил:**старший преподаватель  
Федотов Е. А.

Белгород  
2015

**Цель работы:** изучить принципы и характеристику протокола IPX и разработать программу для приёма/передачи пакетов с использованием библиотеки Winsock.

**Задание:**

1. Разработать программу «Сервер» (на языке программирования C#), которая посылает клиентам некоторое сообщение.
2. Разработать программу «Клиент» (на языке программирования C#), которая ждём от сервера сообщения и затем получает его.
3. Провести анализ функционирования разработанных программ (одновременная работа 2, 3 и т. д. приложений на 2, 3 и т. д. компьютерах ЛВС), сделать выводы.

# Теоретическая часть

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

**Классы сетей IP.** IP-адреса разделяются на 5 классов: A, B, C, D, E. Адреса классов A, B и C делятся на две логические части: номер сети и номер узла.

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

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

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

**Маска подсети.** Поля номеров сети и подсети образуют расширенный сетевой префикс. Для выделения расширенного сетевого префикса используется маска подсети (subnet mask). Таким образом, характеристики IP-адреса полностью задаются собственно IP-адресом и маской подсети.

Стандартные маски:

* Класс A — 255.0.0.0
* Класс B — 255.255.0.0
* Класс C — 255.255.255.0

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

Функции, необходимые для работы с Winsock в этой лабораторной работе:

* WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData) — инициализирует библиотеку Winsock. В случае успеха возвращает 0.
* WSAGetLastError (void) — возвращает код ошибки, возникшей при выполнении последней операции.
* WSACleanup (void) — осуществляет очистку памяти, занимаемой библиотекой Winsock. Функция деинициализирует библиотеку Winsock и возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET\_ERROR. Расширенный код ошибки можно получить при помощи функции WSAGetLastError.
* SOCKET (int af, int type, int protocol) — возвращает либо дескриптор созданного сокета, либо ошибку INVALID \_SOCKET.
* bind (SOCKET s, const struct sockaddr FAR\* name, int namelen) — привязывает сокет к какому-нибудь локальному адресу и порту.
* listen (SOCKET s, int backlog) — переводит сокет в состояние «прослушивания» (для потоковой передачи).
* connect (SOCKET s, const struct sockaddr FAR\* name, int namelen) — используется процессом-клиентом для установления связи с сервером при потоковой передаче. В случае успешного установления соединения connect возвращает 0, иначе SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* accept (SOCKET s, struct sockaddr FAR\* addr, int FAR\* addrlen) — используется для принятия связи на сокет.
* sendto (SOCKET s, const char FAR \* buf, int len, int flags, const struct sockaddr FAR \* to, int tolen) — осуществляет передачу данных с помощью датаграмм.
* send (SOCKET s, const char FAR \* buf, int len, int flags) — осуществляет передачу потоковых данных.
* recvfrom (SOCKET s, char FAR\* buf, int len, int flags, struct sockaddr FAR\* from, int FAR\* fromlen) — осуществляет приём данных с помощью датаграмм.
* recv (SOCKET s, char FAR\* buf, int len, int flags) — осуществляет приём потоковых данных.
* Closesocket (SOCKET s) — служит для закрытия сокета.

# Используемые функции

В этой лабораторной работе использовалась встроенная в .NET реализация Winsock, функционал которой заложен в классах Socket и EndPoint.

Используемые методы класса Socket:

* Bind — связывает объект Socket с локальной конечной точкой;
* ReceiveFrom — принимает датаграмму в буфер данных и сохраняет конечную точку;
* SendTo — посылает данные на указанную точку;
* Connect — создаёт подключение к удалённому хосту;
* Receive — возвращает данные из связанного объекта Socket в приёмный буфер;
* Listen — устанавливает объект Socket в состояние прослушивания;
* Accept — создаёт новый объект Socket для заново созданного подключения;
* Send — передаёт данные в подключённый объект Socket;
* Close — закрывает подключение Socket и освобождает все используемые ресурсы.

# Анализ функционирования программы

Было проведено тестирование программы на одном и на двух компьютерах, с использованием:

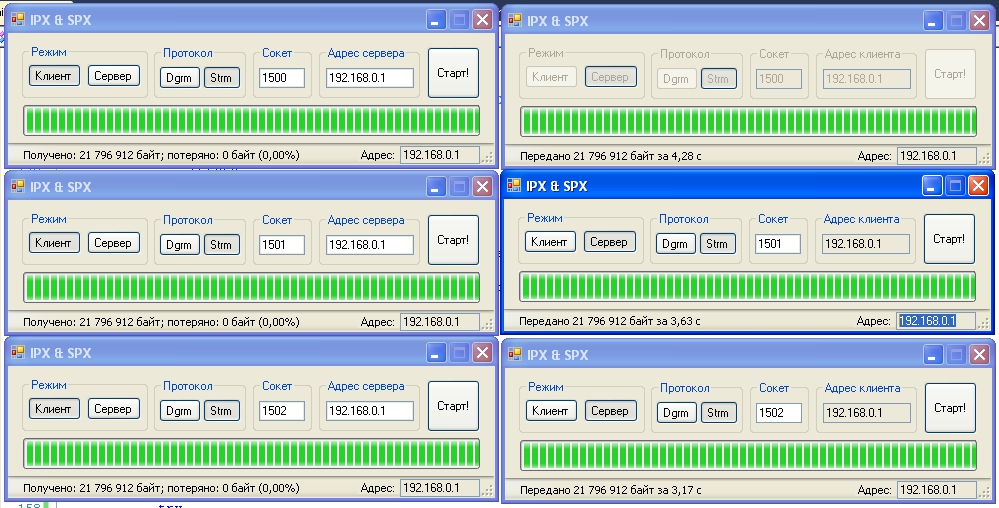
* 1 сервера и 1 клиента;
* 2 серверов и 2 клиентов;
* 3 серверов и 3 клиентов.

Результаты тестирования передачи изображения размером в 21 796 912 байт представлены в таблице ниже:

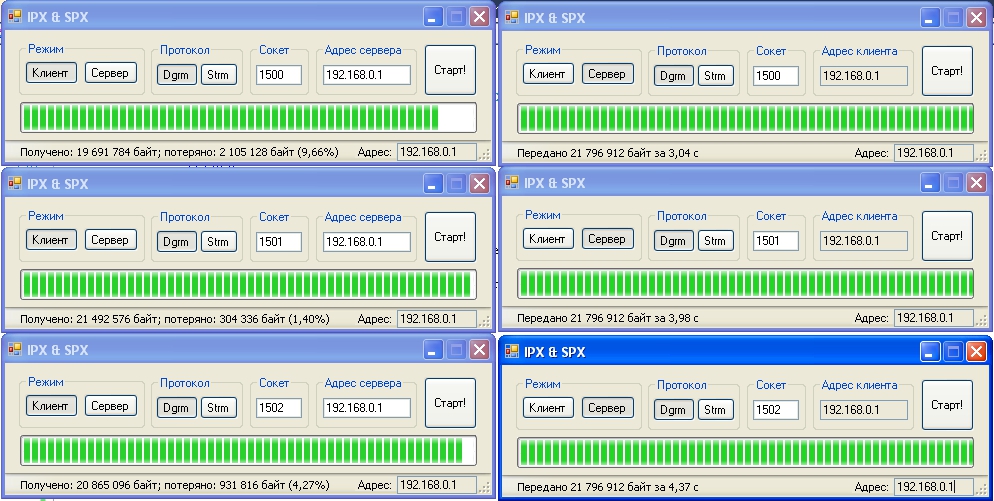
|  |  |  |
| --- | --- | --- |
| **Серверы и клиенты** | **Время** | **Потеряно** |
| **Dgrm, 1 компьютер** | | |
| 1 сервер, 1 клиент | 2,11 с | 337 096 байт, 1,55% |
| 2 сервера, 2 клиента | 3,25 с | 12 016 байт, 0,06% |
| 2,56 с | 984 232 байта, 4,52% |
| 3 сервера, 3 клиента | 3,04 с | 2 105 128 байт, 9,66% |
| 3,98 с | 304 336 байт, 1,40% |
| 4,37 с | 931 816 байт, 4,27% |
| **Dgrm, 2 компьютера** | | |
| 1 сервер, 1 клиент | 12,18 с | 126 928 байт, 0,58% |
| 2 сервера, 2 клиента | 22,63 с | 3 952 байта, 0,02% |
| 20,73 с | 45 280 байт, 0,21% |
| 3 сервера, 3 клиента | 33,34 с | 18 064 байта, 0,08% |
| 34,86 с | 16 552 байта, 0,08% |
| 35,09 с | 14 032 байта, 0,06% |
| **Stream, 1 компьютер** | | |
| 1 сервер, 1 клиент | 2,24 с | 0 байт, 0% |
| 2 сервера, 2 клиента | 2,66 с | 0 байт, 0% |
| 3,14 с | 0 байт, 0% |
| 3 сервера, 3 клиента | 4,28 с | 0 байт, 0% |
| 3,63 с | 0 байт, 0% |
| 3,17 с | 0 байт, 0% |
| **Stream, 2 компьютера** | | |
| 1 сервер, 1 клиент | 9,97 с | 0 байт, 0% |
| 2 сервера, 2 клиента | 16,86 с | 0 байт, 0% |
| 18,21 с | 0 байт, 0% |
| 3 сервера, 3 клиента | 27,06 с | 0 байт, 0% |
| 28,07 с | 0 байт, 0% |
| 28,12 с | 0 байт, 0% |

# Скриншоты приложения

Тестирование потоковой передачи на трёх парах клиент-сервер на одном компьютере:



Тестирование передачи датаграмм на трёх парах клиент-сервер на одном компьютере:



# Код программы

Класс Utils для работы с клиентом и сервером:

1. **using** System;
2. **using** System.Linq;
3. **using** System.Net.NetworkInformation;
4. **using** System.IO;
5. **using** System.Reflection;
6. **using** System.Threading;
7. **using** System.Diagnostics;
8. **using** System.Net;
10. **namespace** Lab3
11. {
12. **class** Utils
13. {
14. **public** **static** **string** GetMAC()
15. {
16. **string** macAddresses = **string**.Empty;
18. **foreach** (NetworkInterface nic **in** NetworkInterface.GetAllNetworkInterfaces())
19. {
20. **if** (nic.OperationalStatus == OperationalStatus.Up)
21. {
22. macAddresses += nic.GetPhysicalAddress().ToString();
23. **break**;
24. }
25. }
27. **return** macAddresses;
28. }
30. **public** **static** **string** GetIPString()
31. {
32. **return** GetIP().ToString();
33. }
35. **public** **static** IPAddress GetIP()
36. {
37. **return** Dns.GetHostEntry(Dns.GetHostName()).AddressList[0];
38. }
40. **public** **static** **void** Starter(**bool** mode, **bool** protocol, **string** ip, **ushort** socket,  
     MainForm form)
41. {
42. **short** packetSize = 512;
43. **short** dataSize = Convert.ToInt16(packetSize - 8);
45. **if** (mode == **true**)
46. {
47. Client client;
49. **if** (protocol == **true**)
50. {
51. client = **new** DgrmClient(**null**, socket);
52. }
53. **else**
54. {
55. client = **new** StrmClient(ip, socket);
56. }
58. var fs = **new** FileStream(Path.GetDirectoryName(  
     Assembly.GetExecutingAssembly().Location) +
59. @"\img-" + socket + ".jpg", FileMode.Create,  
     FileAccess.Write);
61. client.Start();
63. **byte**[] byteArray = **new** **byte**[packetSize];
64. var maxDataBlock = 21; // mb
65. maxDataBlock = (maxDataBlock \* 1024 \* 1024 / dataSize) \* dataSize;
66. **byte**[] saver = **new** **byte**[maxDataBlock];
68. **long** received = 0;
69. **long** length = 0;
70. var dataBlock = 0;
72. byteArray = client.Receive(packetSize);
74. **if** (byteArray.Any(b => b != 0))
75. {
76. **byte**[] tmp = **new** **byte**[8];
77. Array.Copy(byteArray, dataSize, tmp, 0, 8);
78. length = BitConverter.ToInt64(tmp, 0);
80. **while** (byteArray.Any(b => b != 0))
81. {
82. Array.Copy(byteArray, 0, saver, dataBlock, dataSize);
84. dataBlock += dataSize;
86. **if** (dataBlock == maxDataBlock)
87. {
88. fs.Write(saver, 0, dataBlock);
89. dataBlock = 0;
91. Array.Clear(saver, 0, saver.Length);
92. }
94. byteArray = client.Receive(packetSize);
95. received += dataSize;
97. form.UpdateProgess(Convert.ToInt32((received \* 100) / length));
98. }
100. fs.Write(saver, 0, dataBlock);
101. }
103. client.Stop();
105. **double** lost = length - received;
106. lost = lost < 0 ? 0 : lost;
107. received = received > length ? length : received;
109. form.UpdateStatus("Получено: " +  
      String.Format("{0:N0}", received) + " байт; " +
110. "потеряно: " + String.Format("{0:N0}",  
      Convert.ToInt32(lost)) + " байт " +
111. "(" + String.Format(   
      "{0:F2}", lost / length \* 100) + "%)");
113. fs.Close();
114. fs.Dispose();
115. }
116. **else**
117. {
118. Server server;
120. **if** (protocol == **true**)
121. {
122. server = **new** DgrmServer(ip, socket);
123. }
124. **else**
125. {
126. server = **new** StrmServer(**null**, socket);
127. }
129. var fs = **new** FileStream(Path.GetDirectoryName(  
      Assembly.GetExecutingAssembly().Location) +
130. @"\img.jpg", FileMode.Open,  
      FileAccess.Read);
132. **byte**[] byteArray = **new** **byte**[packetSize];
134. **long** length = fs.Length;
135. var fSize = BitConverter.GetBytes(length);
136. **long** total = 0;
138. Stopwatch timer = Stopwatch.StartNew();
140. server.Start();
142. **while** (fs.Read(byteArray, 0, dataSize) > 0)
143. {
144. fSize.CopyTo(byteArray, dataSize);
146. server.Send(byteArray);
148. total += dataSize;
150. form.UpdateProgess(Convert.ToInt32((total \* 100) / length));
152. Array.Clear(byteArray, 0, byteArray.Length);
153. }
155. **for** (**int** i = 0; i < 100; i++)
156. {
157. server.Send(byteArray);
158. Thread.Sleep(1);
159. }
161. server.Stop();
163. timer.Stop();
165. form.UpdateStatus("Передано " +  
      String.Format("{0:N0}", length) + " байт " +
166. "за " + String.Format("{0:N2}",  
      Convert.ToDouble(  
      timer.ElapsedMilliseconds) / 1000) + " с");
168. fs.Close();
169. fs.Dispose();
170. }
171. }
172. }
173. }

Класс для работы с клиентом передачи датаграмм:

1. **using** System.Net.Sockets;
3. **namespace** Lab3
4. {
5. **class** DgrmClient : Client
6. {
7. **private** Socket \_usingsocket;
9. **public** DgrmClient(**string** ip, **ushort** socket) : **base**(ip, socket)
10. {
11. }
13. **public** **override** **void** Start()
14. {
15. \_usingsocket = **new** Socket(AddressFamily.InterNetwork, SocketType.Dgram,  
     ProtocolType.IP);
16. \_usingsocket.Bind(\_endpoint);
17. }
19. **public** **override** **byte**[] Receive(**short** length)
20. {
21. **byte**[] byteArray = **new** **byte**[length];
23. \_usingsocket.ReceiveFrom(byteArray, **ref** \_endpoint);
25. **return** byteArray;
26. }
28. **public** **override** **void** Stop()
29. {
30. \_usingsocket.Close();
31. }
32. }
33. }

Класс для работы с сервером передачи датаграмм:

1. **using** System.Net.Sockets;
3. **namespace** Lab3
4. {
5. **class** DgrmServer : Server
6. {
7. **private** Socket \_usingsocket;
9. **public** DgrmServer(**string** ip, **ushort** socket) : **base**(ip, socket)
10. {
11. }
13. **public** **override** **void** Start()
14. {
15. \_usingsocket = **new** Socket(AddressFamily.InterNetwork, SocketType.Dgram,  
     ProtocolType.IP);
16. }
18. **public** **override** **void** Send(**byte**[] buf)
19. {
20. \_usingsocket.SendTo(buf, \_endpoint);
21. }
23. **public** **override** **void** Stop()
24. {
25. \_usingsocket.Close();
26. }
27. }
28. }

Класс для работы с клиентом потоковой передачи:

1. **using** System.Net.Sockets;
2. **using** System.Net;
4. **namespace** Lab3
5. {
6. **class** StrmClient : Client
7. {
8. **private** Socket \_usingsocket;
9. **private** EndPoint \_localpoint;
11. **public** StrmClient(**string** ip, **ushort** socket) : **base**(ip, socket)
12. {
13. \_localpoint = **new** IPEndPoint(Utils.GetIP(), socket);
14. }
16. **public** **override** **void** Start()
17. {
18. \_usingsocket = **new** Socket(AddressFamily.InterNetwork, SocketType.Stream,  
     ProtocolType.Tcp);
20. \_usingsocket.Connect(\_endpoint);
21. }
23. **public** **override** **byte**[] Receive(**short** length)
24. {
25. **byte**[] byteArray = **new** **byte**[length];
27. \_usingsocket.Receive(byteArray);
29. **return** byteArray;
30. }
32. **public** **override** **void** Stop()
33. {
34. \_usingsocket.Close();
35. }
36. }
37. }

Класс для работы с сервером потоковой передачи:

1. **using** System.Net.Sockets;
3. **namespace** Lab3
4. {
5. **class** StrmServer : Server
6. {
7. **private** Socket \_usingsocket;
8. **private** Socket \_acceptsocket;
10. **public** StrmServer(**string** ip, **ushort** socket) : **base**(ip, socket)
11. {
12. }
14. **public** **override** **void** Start()
15. {
16. \_usingsocket = **new** Socket(AddressFamily.InterNetwork, SocketType.Stream,  
     ProtocolType.Tcp);
18. \_usingsocket.Bind(\_endpoint);
19. \_usingsocket.Listen(10);
20. \_acceptsocket = \_usingsocket.Accept();
21. }
23. **public** **override** **void** Send(**byte**[] buf)
24. {
25. **try**
26. {
27. \_acceptsocket.Send(buf);
28. }
29. **catch**
30. {
31. **return**;
32. }
33. }
35. **public** **override** **void** Stop()
36. {
37. \_usingsocket.Close();
38. \_acceptsocket.Close();
39. }
40. }
41. }

# Ответы на вопросы

1. **Сущность протокола IP**Internet Protocol или IP (англ. internet protocol — межсетевой протокол) — маршрутизируемый сетевой протокол сетевого уровня семейства TCP/IP.  
     
   Протокол IP используется для негарантированной доставки данных, разделяемых на так называемые пакеты от одного узла сети к другому. Это означает, что на уровне этого протокола (третий уровень сетевой модели OSI) не даётся гарантий надёжной доставки пакета до адресата. В частности, пакеты могут прийти не в том порядке, в котором были отправлены, продублироваться (когда приходят две копии одного пакета — в реальности это бывает крайне редко), оказаться повреждёнными (обычно повреждённые пакеты уничтожаются) или не прибыть вовсе. Гарантии безошибочной доставки пакетов дают протоколы более высокого (транспортного) уровня сетевой модели OSI, например, TCP — который использует IP в качестве транспорта.  
     
   Обычно в сетях используется IP четвёртой версии, также известный как IPv4. В протоколе IP этой версии каждому узлу сети ставится в соответствие IP-адрес длиной 4 октета (1 октет состоит из 8 бит). При этом компьютеры в подсетях объединяются общими начальными битами адреса. Количество этих бит, общее для данной подсети, называется маской подсети (ранее использовалось деление пространства адресов по классам: A, B, C; класс сети определяется диапазоном значений старшего октета и определяет число адресуемых узлов в данной сети).  
     
   IP-пакет представляет собой форматированный блок информации, передаваемый по вычислительной сети. Соединения вычислительных сетей, которые не поддерживают пакеты, такие как традиционные соединения типа «точка-точка» в телекоммуникациях, просто передают данные в виде последовательности байтов, символов или битов. При использовании пакетного форматирования сеть может передавать длинные сообщения более надёжно и эффективно.
2. **Опишите структуру IP-адресов в классах A, C, B, D, E**
3. **Какие IP-адреса являются зарезервированными для специального использования?**
   1. Если все биты IP-адреса установлены в нуль, то он обозначает адрес данного устройства.
   2. Если в поле номера сети стоят нули, то считается, что получатель принадлежит той же самой сети, что и отправитель.
   3. Если все биты IP-адреса установлены в единицу, то пакет с таким адресом должен рассылаться всем узлам, находящимся в той же сети, что и отправитель. Такая рассылка называется ограниченным широковещательным сообщением.
   4. Если все биты номера узла установлены в нуль, то пакет предназначен для данной сети.
   5. Если все биты в поле номера узла установлены в единицу, то пакет рассылается всем узлам сети с данным номером сети. Такая рассылка называется широковещательным сообщением.
   6. Если первый октет адреса равен 127, то адрес обозначает тот же самый узел. Такой адрес используется для взаимодействия процессов на одной и той же машине (например, для целей тестирования). Этот адрес имеет название возвратного.
4. **Что представляет собой маска подсети?**Маска подсети — это 32-разрядное двоичное число, в разрядах расширенного префикса содержащая единицу; в остальных разрядах находится ноль. Расширенный сетевой префикс получается побитным сложением по модулю два (операция XOR) IP-адреса и маски подсети.
5. **Как создать сокет для работы с протоколом IP?**В C#:  
   \_usingsocket = new Socket(AddressFamily.InterNetwork,  
    SocketType.Dgram, ProtocolType.IP);
6. **Какие функции Winsocket необходимо вызвать для вывода на экран IP-адреса комьютера?**gethostbyname и getsocketname
7. **Разбейте сеть 10.10.0.0/15 на 8 частей. Запишите диапазоны доступных адресов в каждой из получившихся сетей.**Запишем адрес сети и маску в двоичном виде:

01000110 01000110 00000000 00000000 (10.10.0.0)

11111111 11111110 00000000 00000000 (255.254.0.0)

Увеличим длину маски на 3 бита (т. к. 8 — это 2 в степени 3):

11111111 11111111 11000000 00000000 (255.255.192.0, длина маски - 18).

Наложим новую маску на сеть:

01000110 01000110 00000000 00000000 (10.10.0.0)

11111111 11111111 11000000 00000000 (255.255.192.0)

В образовании новой сети будут участвовать восьмой бит второго октета(шестнадцатый слева) и первый и второй бит третьего октета (семнадцатый и восемнадцатый биты слева). Итого получим 8 сетей:

00001010 00001010 00000000 00000000 (10.10.0.0/18)

00001010 00001010 01000000 00000000 (10.10.64.0/18)

00001010 00001010 10000000 00000000 (10.10.128.0/18)

00001010 00001010 11000000 00000000 (10.10.192.0/18)

00001010 00001011 00000000 00000000 (10.11.0.0/18)

00001010 00001011 01000000 00000000 (10.11.64.0/18)

00001010 00001011 10000000 00000000 (10.11.128.0/18)

00001010 00001011 11000000 00000000 (10.11.192.0/18)

Доступные в каждой из сетей адреса будут следующими:

10.10.0.0/18: 10.10.0.1 — 10.10.63.254

10.10.64.0/18: 10.10.64.1 — 10.10.127.254

10.10.128.0/18: 10.10.128.1 — 10.10.191.254

10.10.192.0/18: 10.10.192.1 — 10.10.255.254

10.11.0.0/18: 10.11.0.1 — 10.11.63.254

10.11.64.0/18: 10.11.64.1 — 10.11.127.254

10.11.128.0/18: 10.11.128.1 — 10.11.191.254

10.11.192.0/18: 10.11.192.1 — 10.11.255.254

# Блок-схемы работы программы