Белгородский государственный технологический университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №1 по теме: «Протокол сетевого уровня IPX»

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

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

Белгород  
2015

**Цель работы:** изучить протокол сетевого уровня IPX, основные функции API драйвера IPX и разработать программу для приёма/передачи данных.

**Задание:**

1. Разработать программу «Сервер», которая посылает клиентам сети файл с использованием протокола IPX в среде DOS на языке программирования Pascal или C.
2. Разработать программу «Клиент», которая принимает от сервера файл на языке программирования Pascal или C.
3. Провести анализ функционирования разработанных программ при передаче файла в формате .jpg размером не менее 1 Мб (одновременная работа 2-х, 3-х и т. д. приложений на 2-х, 3-х и т. д. компьютерах ЛВС), сделать выводы.

# Теоретическая часть

Протокол IPX — это протокол сетевого уровня модели взаимодействия открытых систем (OSI) реализующий передачу пакетов (сообщений) между станциями сети на уровне датаграмм. Датаграмма — это сообщение, доставка которого получателю не гарантируется. Следовательно, для обеспечения надёжной работы нужно предусмотреть схему уведомления других станций о том, что переданные ими пакеты успешно приняты и обработаны. Более того, последовательность отправления пакетов передающим узлом может отличаться от последовательности приёма этих пакетов, что также необходимо учитывать.

В процессе обмена сообщениями на уровне сеанса связи участвуют только две станции сети. На уровне датаграмм есть возможность посылать сообщение одновременно всем станциям сети.

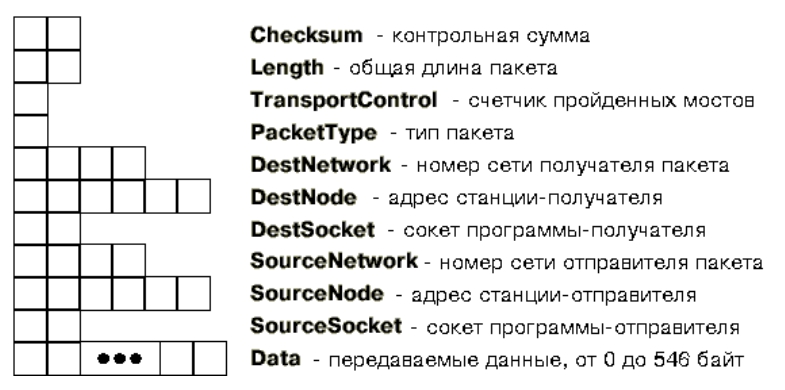
Система адресов, используемая в протоколе IPX, представлена несколькими компонентами: это номер сети, адрес станции в сети и идентификатор программы на рабочей станции.

**Номер сети** — это номер сегмента сети, определяемого системным администратором. Если в общей сети есть мосты, каждая отдельная сеть, подключённая через мост, должна иметь свой, уникальный номер сети.

**Адрес станции** — это число, которое является уникальным для каждой рабочей станции. При использовании адаптеров Ethernet уникальность обеспечивается изготовителем сетевого адаптера. Специальный адрес FFFFFFFFFFFFh используется для рассылки данных всем станциям данной сети одновременно.

**Идентификатор программы на рабочей станции (сокет)** — число, которое используется для адресации определённой программы, работающей на компьютере. В среде мультизадачных операционных систем, на каждой рабочей станции в сети одновременно может быть запущено несколько приложений. Для того, чтобы послать данные конкретной программе, используется идентификация программ при помощи сокетов. Каждая программа, желающая принимать или передавать данные по сети, должна получить свой, уникальный для данной рабочей станции, идентификатор — сокет.

Формат передаваемых с использованием протокола IPX по сети пакетов представлен на рисунке ниже.



Все поля, кроме поля Data, представляют собой заголовок пакета. В заголовке располагаются: адрес назначения, обратный адрес и некоторая служебная информация. Все поля заголовка содержат значения в перевёрнутом формате, т. е. по младшему адресу записывается младший байт данных.

Поле **Checksum** предназначено для хранения контрольной суммы передаваемых пакетов. При формировании пакетов не нужно заботиться о содержимом этого поля, так как проверка данных по контрольной сумме выполняется драйвером сетевого адаптера.

Поле **Length** определяет общий размер пакета. Длина заголовка фиксирована и равна 30 байт. Размер передаваемых в поле Data данных может составлять от 0 до 546 байт. При формировании пакетов не нужно проставлять длину пакета в поле Length, протокол IPX делает это сам.

Поле **TransportControl** является счётчиком мостов, которые проходит пакет на своём пути от передающей станции к принимающей. Каждый раз, когда пакет проходит через мост, значение этого счётчика увеличивается на единицу. IPX перед передачей пакета сбрасывает содержимое этого поля в нуль. Так как IPX сам следит за содержимым этого поля, при формировании пакетов не нужно изменять или устанавливать его в какое-либо состояние.

Поле **PacketType** определяет тип передаваемого пакета. Для IPX следует установить значение равное 4.

Поле **DestNetwork** определяет номер сети, в которую передаётся пакет.

Поле **DestNode** определяет адрес рабочей станции, которой предназначен пакет.

Поле **DestSocket** предназначено для адресации программы, запущенной на рабочей станции, которая должна принять пакет.

Поля **SourceNetwork**, **SourceNode** и **SourceSocket** содержат соответственно номер сети, из которой посылается пакет, адрес передающей станции и сокет программы, передающей пакет.

Поле **Data** содержит передаваемые данные.

Прикладные программы все свои запросы на приём и передачу пакетов направляют драйверу IPX, который, в свою очередь, обращается к драйверу сетевого адаптера. Для приёма или передачи пакета прикладная программа должна подготовить пакет данных, сформировав его заголовок, и построить так называемый блок управления событием ECB (Event Control Block). В блоке ECB задаётся адресная информация для передачи пакета, адрес передаваемого пакета в оперативной памяти и некоторая другая информация. Подготовив блок ECB, прикладная программа передаёт его адрес соответствующей функции IPX для выполнения операции приёма или передачи пакета.

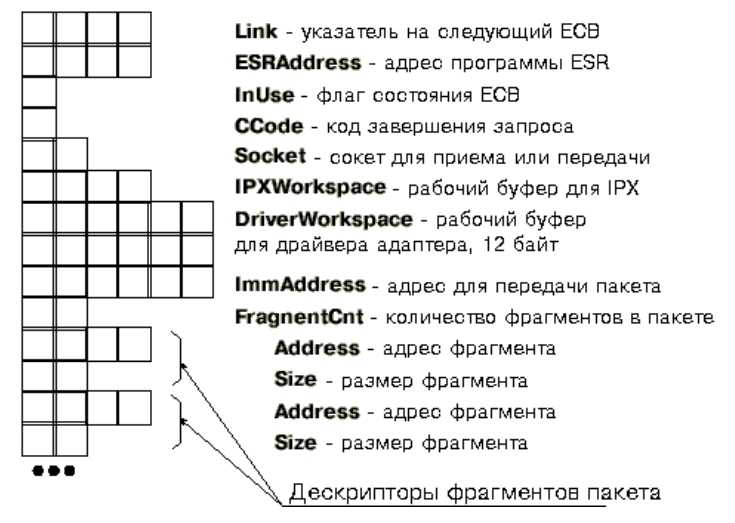
Функции IPX, принимающие или передающие пакет, не выполняют ожидания завершения операции, а сразу возвращают управление вызвавшей их программе. Приём или передача выполняются сетевым адаптером автономно и асинхронно по отношению к программе, вызвавшей функцию IPX для передачи данных. После того, как операция передачи данных завершилась, в соответствующем поле блока ECB устанавливается признак завершения операции. Программа может периодически проверять ECB для обнаружения этого признака.

Кроме того, в блоке ECB следует указать адрес процедуры, которая будет вызвана при завершении выполнения операции передачи данных. Этот способ предпочтительнее, так как прикладная программа не будет тратить время на периодическую проверку блока ECB.

Формат блока ECB представлен на рисунке далее.

Блок ECB состоит из фиксированной части размером 36 байт и массива дескрипторов, описывающих отдельные фрагменты передаваемого или принимаемого пакета данных.

Поле **Link** предназначено для организации списков, состоящих из блоков ECB. Драйвер IPX использует это поле для объединения переданных ему блоков ECB в списки, записывая в него полный адрес в формате [сегмент : смещение]. После того, как IPX выполнит данную ему команду и закончит все операции над блоком ECB, программа может распоряжаться полем Link по своему усмотрению. В частности, она может использовать это поле для организации списков или очередей свободных, или готовых для чтения блоков ECB.



Поле **ESRAddress** содержит полный адрес программного модуля (в формате [сегмент : смещение]), который получает управление при завершении процесса чтения или передачи пакета IPX. Этот модуль называется программой обслуживания события ESR (Event Service Routine). Если программа не использует ESR, она должна записать в поле ESRAddress нулевое значение. В этом случае о завершении выполнения операции чтения или передачи можно узнать по изменению содержимого поля InUse.

Поле **InUse** может служить индикатором завершения операции приёма или передачи пакета. Перед тем как вызвать функцию IPX, программа записывает в поле InUse нулевое значение. Пока операция передачи данных, связанная с данным ECB, не завершилась, поле InUse содержит следующие значения:

* FFh — ECB используется для передачи пакета данных;
* FEh — ECB используется для приёма пакета данных, предназначенного программе с конкретным сокетом;
* FDh — ECB используется функциями асинхронного управления событиями AES (Asynchronous Event Sheduler), ECB находится в состоянии ожидания истечения заданного временного интервала;
* FBh — пакет данных принят или передан, но ECB находится во внутренней очереди IPX в ожидании завершения обработки.

Программа может постоянно опрашивать поле InUse, ожидая завершения процесса передачи или приёма данных. Как только в этом поле окажется нулевое значение, программа считает, что запрошенная функция выполнена. Результат выполнения можно получить в поле **CCode**, где после выполнения функции IPX содержится код результата выполнения.

Если с данным ECB была связана команда приема пакета, в поле CCode могут находиться следующие значения:

* 00 — пакет был принят без ошибок;
* FFh — указанный в ECB сокет не был предварительно открыт программой;
* FDh — переполнение пакета: либо поле количества фрагментов в пакете FragmentCnt равно нулю, либо буферы, описанные дескрипторами фрагментов, имеют недостаточный размер для записи принятого пакета;
* FCh - запрос на прием данного пакета был отменен специальной функцией драйвера IPX.

Если ECB использовался для передачи пакета, в поле CCode после завершения передачи могут находиться следующие значения:

* 00 — пакет был передан без ошибок (но это не означает, что пакет был доставлен по назначению и успешно принят станцией-адресатом);
* FFh — пакет невозможно передать физически из-за неисправности в сетевом адаптере или в сети;
* FEh — пакет невозможно доставить по назначению, так как станция с указанным адресом не существует или неисправна;
* FDh — пакет сбойный, т.е. либо имеет длину меньше 30 байт, либо первый фрагмент пакета по размеру меньше размера стандартного заголовка пакета IPX, либо поле количества фрагментов в пакете FragmentCnt равно нулю;
* FCh — запрос на передачу данного пакета был отменен специальной функцией драйвера IPX.

Поле **Socket** содержит номер сокета, связанный с данным ECB. Если ECB используется для приёма, это поле содержит номер сокета, на котором выполняется приём пакета. Если же ECB используется для передачи, это поле содержит номер сокета передающей программы.

Поле **IPXWorkspace** зарезервировано для использования драйвером IPX. Приложение не должно инициализировать или изменять содержимое этого поля, пока обработка ECB не завершена.

Поле **DriverWorkspace** зарезервировано для использования драйвером сетевого адаптера. Программа не должна инициализировать или изменять содержимое этого поля, пока обработка ECB не завершена.

Поле **ImmAddress** (Immediate Address — непосредственный адрес) содержит адрес узла в сети, в который будет направлен пакет. Если пакет передаётся в пределах одной сети, поле ImmAddress будет содержать адрес станции-получателя (такой же, как и в заголовке пакета IPX). Если же пакет предназначен для другой сети и будет проходить через мост, поле ImmAddressбудет содержать адрес этого моста в сети, из которой передаётся пакет.

Поле **FragmentCnt** содержит количество фрагментов, на которые нужно разбить принятый пакет, или из которых необходимо собрать передаваемый пакет. Механизм фрагментации позволяет избежать пересылок данных или непроизводительных потерь памяти. Можно указать отдельные буферы для приёма данных и заголовка пакета. Если принимаемые данные имеют какую-либо структуру, можно рассредоточить отдельные блоки по соответствующим буферам. Значение, записанное в поле FragmentCnt, не должно быть равно нулю. Если в этом поле записано значение 1, весь пакет вместе с заголовком записывается в один общий буфер.

Далее располагаются дескрипторы фрагментов, состоящие из указателя в формате [сегмент : смещение] на фрагмент **Address** и поля размера фрагмента **Size**.

Если программе нужно разбить принятый пакет на несколько частей, то нужно установить в поле FragmentCnt значение, равное количеству требуемых фрагментов. А для каждого фрагмента необходимо создать дескриптор, в котором указать адрес буфера и размер фрагмента. Аналогичные действия выполняются и при сборке пакета перед передачей из нескольких фрагментов.

# Алгоритм работы программы

Сервер:

1. Проверка на наличие IPX.
2. Запрос у пользователя номера сокета. Открытие сокета, если номер был введён, иначе открытие по умолчанию — 20480.
3. Запрос у пользователя названия JPEG-изображения. Если название не было введено, используется по умолчанию «in.jpg».
4. Открытие файла средствами Паскаля, установка минимального блока для считывания в 1 байт.
5. Подсчёт размера файла.
6. Циклическое считывание 503 (максимум) блоков, добавление к ним 8 байт, содержащих полный размер файла и 1 байта, содержащего флаг: 1 — пакет с файлом, 0 — пустой пакет. Отправка блоков клиентам.
7. После отправки файла происходит посыл ещё 100 дополнительных пустых пактов, содержащих установленный флаг, равный 0.
8. Закрытие сокета и файла.

Клиент:

1. Проверка на наличие IPX.
2. Запрос у пользователя номера сокета. Открытие сокета, если номер был введён, иначе открытие по умолчанию — 20480.
3. Запрос у пользователя названия создаваемого JPEG-изображения. Если название не было введено, используется по умолчанию «out.jpg».
4. Создание и открытие файла средствами Паскаля, установка минимального блока для считывания в 1 байт.
5. Циклическое получение сообщений. Проверка флага, если он равен 0, то получение прерывается. Если равен 1, то происходит запись принятых данных в файл. При первой записи также происходит расчёт полного размера файла (с помощью переданных данных).
6. Подсчёт потерянных данных.
7. Закрытие сокета и файла.

# Анализ функционирования программы

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

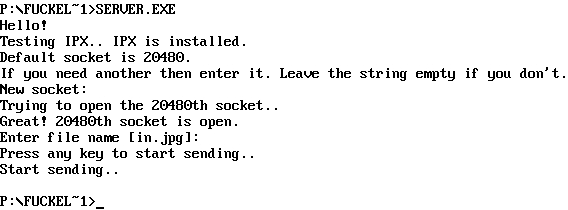
* 1 сервера и 1 клиента;
* 2 серверов и 2 клиентов;
* 3 серверов и 3 клиентов.

Результаты тестирования передачи изображения размером в 1 135 595 байт представлены в таблице ниже:

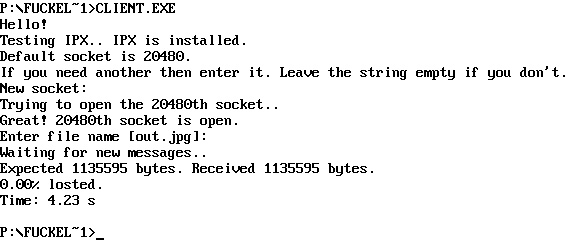
|  |  |  |
| --- | --- | --- |
| **Серверы и клиенты** | **Время** | **Потеряно** |
| **1 компьютер** | | |
| 1 сервер, 1 клиент | 4,23 с | 0 байт, 0% |
| 2 сервера, 2 клиента | 4,23 с | 0 байт, 0% |
| 4,23 с | 0 байт, 0% |
| 3 сервера, 3 клиента | 4,18 с | 0 байт, 0% |
| 4,34 с | 0 байт, 0% |
| 4,18 с | 0 байт, 0% |
| **3 компьютера** | | |
| 1 сервер, 1 клиент |  |  |
| 2 сервера, 2 клиента |  |  |
|  |  |
| 3 сервера, 3 клиента |  |  |
|  |  |
|  |  |

# Скриншоты приложения

**Сервер:**



**Клиент:**



# Код программы

Модуль для работы с IPX:

1. UNIT UIPX;
3. {-----------------------------------------------------------------}
4. INTERFACE
5. {-----------------------------------------------------------------}
7. **type**
8. TFullAddress = **record**
9. Network     : **array** [0..3] **of** **byte**;
10. Node        : **array** [0..5] **of** **byte**;
11. Socket      : **word**;
12. **end**;
14. PFullAddress = ^TFullAddress;
16. TIPXHeader = **record**
17. Checksum            : **word**;
18. Length              : **word**;
19. TransportControl    : **byte**;
20. PacketType          : **byte**;
21. Destination         : TFullAddress;
22. Source              : TFullAddress;
23. **end**;
25. PIPXHeader = ^TIPXHeader;
27. TECB = **record**
28. Link            : **pointer**;
29. ESRAddress      : **pointer**;
30. InUse           : **byte**;
31. CCode           : **byte**;
32. Socket          : **word**;
33. IPXWorkspace    : **array** [0..3] **of** **byte**;
34. DriverWorkspace : **array** [0..11] **of** **byte**;
35. ImmAddress      : **array** [0..5] **of** **byte**;
36. FragmentCount   : **word**;
37. F1Address       : **pointer**;
38. F1Size          : **word**;
39. F2Address       : **pointer**;
40. F2Size          : **word**;
41. **end**;
43. PECB = ^TECB;
45. { I/O type}
46. TIO = **record**
47. IPXHeader   : TIPXHeader;
48. ECB         : TECB;
49. Data        : **array** [0..511] **of** **byte**;
50. **end**;
52. PIO = ^TIO;
54. **var**
55. fullAddress : TFullAddress;

58. **function** AllocIO: **boolean**;
59. { Called automatically on startup and allocated memory of IO }
61. **function** FreeIO: **boolean**;
62. { Free memory of IO (it needs to call it in the end) }
64. **function** IsIPXInstalled: **boolean**;
65. { Detects IPX }
67. **procedure** IPXGetLocalTarget(**var** **addr**: TFullAddress);
68. { Returns full network address }
70. **function** IPXOpenSocket(**var** socket: **word**): **byte**;
71. { Opens the `socket` }
73. **procedure** IPXCloseSocket(socket: **word**);
74. { Closes the `socket` }
76. **procedure** IPXRelinquishControl;
77. { Allocate CPU time for IPX. It needs to call it in waiting loop }
79. **procedure** IPXSendData(**var** data);
80. { Sends broadcasting 512-bytes packets }
82. **procedure** IPXPrepareForReceiving;
83. { Prepears receiving one data block }
85. **function** IPXReceiveData(**var** data): **boolean**;
86. { Returns true when reveives data }
88. {-----------------------------------------------------------------}
89. IMPLEMENTATION
90. {-----------------------------------------------------------------}
92. **uses** Dos;
94. **var**
95. IOBuf   : PIO;
97. **function** AllocIO: **boolean**;
98. **begin**
100. getmem(IOBuf, sizeof(TIO));
102. AllocIO := **true**;
104. **end**;
106. **function** FreeIO: **boolean**;
107. **begin**
109. freemem(IOBuf, sizeof(TIO));
111. FreeIO := **true**;
113. **end**;
115. **function** IsIPXInstalled: **boolean**; assembler;
116. **asm**
118. MOV AX, 7A00h
119. INT 2Fh
120. AND AL, 1
122. **end**;
124. **procedure** IPXGetLocalTarget(**var** **addr**: TFullAddress);
125. **var**
126. r: registers;
128. **begin**
130. r.BX := $09;
131. r.ES := seg(**addr**);
132. r.SI := ofs(**addr**);
133. intr($7A, r);
135. **end**;
137. **function** IPXOpenSocket(**var** socket: **word**): **byte**;
138. **var**
139. r: registers;
141. **begin**
143. IPXCloseSocket(socket);
145. r.BX := $00;
146. r.AL := $FF;
147. r.DX := swap(socket);
149. intr($7A, r);
151. socket := swap(r.DX);
153. IPXOpenSocket := r.AL;
155. **end**;
157. **procedure** IPXCloseSocket(socket: **word**);
158. **var**
159. r: registers;
161. **begin**
163. r.BX := $01;
164. r.DX := swap(socket);
165. intr($7A, r);
167. **end**;
169. **function** IPXListenForPacket(**var** ECB: TECB): **byte**;
170. { Not for interface. It needs for start of receiving packets }
171. **var**
172. r: registers;
174. **begin**
176. r.BX := $04;
177. r.ES := seg(ECB);
178. r.SI := ofs(ECB);
179. intr($7A, r);
181. IPXListenForPacket := r.AL;
183. **end**;
185. **function** IPXSendPacket(**var** ECB: TECB): **byte**;
186. { Not for interdace. It needs for preparing of ECB for sending }
187. **var**
188. r: registers;
190. **begin**
192. r.BX := $03;
193. r.ES := seg(ECB);
194. r.SI := ofs(ECB);
195. intr($7A, r);
197. **end**;
199. **procedure** IPXRelinquishControl;
200. **var**
201. r: registers;
203. **begin**
205. r.BX := $A;
206. intr($7A, r);
208. **end**;
210. **procedure** IPXSendData(**var** data);
211. **var**
212. p: PIO;
213. node: **array** [0..5] **of** **byte**;
215. **begin**
217. p := IOBuf;
219. fillchar(node, 6, $FF); { broadcast }
221. **with** p^.IPXHeader **do**
222. **begin**
223. Checksum            := 0;
224. Length              := sizeof(TIPXHeader) + 512;
225. TransportControl    := 0;
226. PacketType          := 4;
227. Destination.Network := fullAddress.Network;
228. Destination.Socket  := fullAddress.Socket;
229. move(node, Destination.Node, sizeof(fullAddress.Node));
230. move(fullAddress, Source, sizeof(TFullAddress));
231. **end**;
233. **with** p^.ECB **do**
234. **begin**
235. Link            := **nil**;
236. ESRAddress      := **nil**;
237. Socket          := fullAddress.Socket;
238. FragmentCount   := 2;
239. F1Address       := @p^.IPXHeader;
240. F1Size          := sizeof(TIPXHeader);
241. F2Address       := @p^.Data;
242. F2Size          := 512;
243. fillchar(IPXWorkspace, 4, 0);
244. fillchar(DriverWorkspace, 12, 0);
245. move(node, ImmAddress, sizeof(fullAddress.Node));
246. **end**;
248. move(data, p^.Data, 512);
249. IPXSendPacket(p^.ECB);
251. **end**;
253. **procedure** IPXPrepareForReceiving;
254. **var**
255. p: PIO;
257. **begin**
259. p := IOBuf;
261. **with** p^.ECB **do**
262. **begin**
263. Link            := **nil**;
264. ESRAddress      := **nil**;
265. Socket          := fullAddress.Socket;
266. FragmentCount   := 2;
267. F1Address       := @p^.IPXHeader;
268. F1Size          := sizeof(TIPXHeader);
269. F2Address       := @p^.Data;
270. F2Size          := 512;
271. fillchar(IPXWorkspace, 4, 0);
272. fillchar(DriverWorkspace, 12, 0);
273. fillchar(ImmAddress, 6, $FF);
274. **end**;
276. IPXListenForPacket(p^.ECB);
278. **end**;
280. **function** IPXReceiveData(**var** data): **boolean**;
281. **var**
282. p: PIO;
284. **begin**
286. p := IOBuf;
288. **if** (p^.ECB.InUse = 0) **then**
289. **begin**
290. move(p^.Data, data, 512);
291. IPXPrepareForReceiving;
293. IPXReceiveData := **true**;
294. **end**
295. **else**
296. IPXReceiveData := **false**;
298. **end**;
300. **begin**
302. AllocIO;
304. **end**.

Сервер:

1. **program** IPXServer;
3. **uses** UIPX, CRT;
5. **const**
6. DEFAULT\_SOCKET = $5000;
7. PACKAGES = 503;
9. **var**
10. key     : **char**;
11. tmp     : **string**;
12. socket  : **word**;
13. error   : **integer**;
14. f       : **file**;
15. fname   : **string**;
16. block   : **array** [0..511] **of** **byte**;
17. fsize   : **longint**;
18. arsize  : **array** [0..7] **of** **byte** absolute fsize;
19. i       : **longint**;
20. j       : **byte**;
22. **label** theEnd;
24. **begin**
26. **writeln**('Hello!');
27. **write**('Testing IPX.. ');
29. **if** (**not** IsIPXInstalled) **then**
30. **begin**
31. **writeln**('IPX is not installed. Bye!');
32. **goto** theEnd;
33. **end**
34. **else**
35. **writeln**('IPX is installed.');
37. **writeln**('Default socket is 20480.');
38. **writeln**('If you need another then enter it. Leave the string empty if you don''t.');
39. **write**('New socket: ');
40. readln(tmp);
42. **if** (length(tmp) = 0) **then**
43. socket := DEFAULT\_SOCKET
44. **else**
45. **begin**
46. **val**(tmp, socket, error);
47. **if** (error <> 0) **then**
48. **begin**
49. **writeln**('You didn''t enter a number');
50. **goto** theEnd;
51. **end**;
52. **end**;
54. **writeln**('Trying to open the ', socket, 'th socket..');
55. **if** (IPXOpenSocket(socket) <> 0) **then**
56. **begin**
57. **writeln**('Coundn''t open the ', socket, 'th socket!');
58. **goto** theEnd;
59. **end**
60. **else**
61. **writeln**('Great! ', socket, 'th socket is open.');
63. fullAddress.Socket := swap(socket);
65. **write**('Enter file name [in.jpg]: ');
66. readln(tmp);
68. **if** (length(tmp) = 0) **then**
69. fname := 'in.jpg'
70. **else**
71. fname := tmp;
73. assign(f, fname);
74. reset(f, 1);
76. fsize := filesize(f);
78. i := 0;
80. **write**('Press any key to start sending..');
81. readln;
83. **writeln**('Start sending..');
84. key := #$FF;
86. **repeat**
88. IPXRelinquishControl;
90. **if** ((i + PACKAGES) >= fsize) **then**
91. blockread(f, block, fsize - i)
92. **else**
93. blockread(f, block, PACKAGES);
95. **for** j := 0 **to** 7 **do**
96. **begin**
98. block[PACKAGES + j] := arsize[j];
100. **end**;
102. block[511] := 1;
104. IPXSendData(block);
105. delay(1);
107. **if** keypressed **then**
108. key := readkey;
110. **if** (key = #27) **then** break;
112. inc(i, PACKAGES);
113. seek(f, i);
115. **until** (i >= fsize);
117. **for** i := 1 **to** 100 **do**
118. **begin**
120. block[511] := 0;
121. IPXSendData(block);
122. delay(5);
124. **end**;
126. IPXRelinquishControl;
128. IPXCloseSocket(socket);
130. close(f);
132. theEnd:
133. FreeIO;
135. **end**.

Клиент:

1. **program** IPXClient;
3. **uses** UIPX, CRT, DOS;
5. **const**
6. DEFAULT\_SOCKET = $5000;
7. PACKAGES = 503;
9. **function** GetTimestamp: **longint**;
10. **var**
11. hour, min, sec, hund: **word**;
13. **begin**
15. GetTime(hour, min, sec, hund);
16. GetTimestamp := hour \* 360000 + min \* 6000 + sec \* 100 + hund;
18. **end**;
20. **var**
21. socket  : **word**;
22. tmp     : **string**;
23. error   : **integer**;
24. key     : **char**;
25. f       : **file**;
26. fname   : **string**;
27. block   : **array** [0..511] **of** **byte**;
28. fsize   : **longint**;
29. result  : **real**;
30. arsize  : **array** [0..7] **of** **byte** absolute fsize;
31. i       : **longint**;
32. j       : **byte**;
33. counted : **boolean**;
34. timer   : **longint**;
36. **label** theEnd;
38. **begin**
40. **writeln**('Hello!');
41. **write**('Testing IPX.. ');
43. **if** (**not** IsIPXInstalled) **then**
44. **begin**
45. **writeln**('IPX is not installed. Bye!');
46. **goto** theEnd;
47. **end**
48. **else**
49. **writeln**('IPX is installed.');
51. **writeln**('Default socket is 20480.');
52. **writeln**('If you need another then enter it. Leave the string empty if you don''t.');
53. **write**('New socket: ');
54. readln(tmp);
56. **if** (length(tmp) = 0) **then**
57. socket := DEFAULT\_SOCKET
58. **else**
59. **begin**
60. **val**(tmp, socket, error);
61. **if** (error <> 0) **then**
62. **begin**
63. **writeln**('You didn''t enter a number');
64. **goto** theEnd;
65. **end**;
66. **end**;
68. **writeln**('Trying to open the ', socket, 'th socket..');
69. **if** (IPXOpenSocket(socket) <> 0) **then**
70. **begin**
71. **writeln**('Coundn''t open the ', socket, 'th socket!');
72. **goto** theEnd;
73. **end**
74. **else**
75. **writeln**('Great! ', socket, 'th socket is open.');
77. fullAddress.Socket := swap(socket);
79. IPXPrepareForReceiving;
81. **write**('Enter file name [out.jpg]: ');
82. readln(tmp);
84. **if** (length(tmp) = 0) **then**
85. fname := 'out.jpg'
86. **else**
87. fname := tmp;
89. assign(f, fname);
90. rewrite(f, 1);
92. **writeln**('Waiting for new messages..');
94. counted := **false**;
95. i := 0;
96. fsize := 1;
97. key := #$FF;
99. **repeat**
101. IPXRelinquishControl;
102. **if** (IPXReceiveData(block)) **then**
103. **begin**
104. **if** (**not** counted) **then**
105. timer := GetTimestamp;
107. **if** (block[511] = 0) **then** break;
109. **if** (counted **and** ((i + PACKAGES) >= fsize)) **then**
110. blockwrite(f, block, fsize - i)
111. **else**
112. blockwrite(f, block, PACKAGES);
113. inc(i, PACKAGES);
115. **if** (**not** counted) **then**
116. **begin**
117. **for** j := 0 **to** 7 **do**
118. arsize[j] := block[PACKAGES + j];
120. counted := **true**;
121. **end**;
122. **end**;
124. **if** keypressed **then**
125. key := readkey;
127. **if** (key = #27) **then** break;
129. **until** (i >= fsize);
131. timer := GetTimestamp - timer;
133. result := filesize(f);
135. **writeln**('Expected ', fsize, ' bytes. Received ', trunc(result), ' bytes.');
136. **writeln**(((fsize - result) / fsize \* 100):0:2, '% losted.');
137. **writeln**('Time: ', (timer / 100):0:2, ' s');
139. IPXRelinquishControl;
141. IPXCloseSocket(socket);
143. close(f);
145. theEnd:
146. FreeIO;
148. **end**.