Шестая межрегиональная научная конференция
«Шаг в будущее, Москва»

регистрационный номер

Информатика и системы управления

Система автоматизированного оповещения пользователей в локальной сети

Автор: **Давыдова Мария Владимировна**
лицей №1580, 11 класс

Руководитель: **Дединский Илья Рудольфович**
преподаватель программирования лицея №1580

Москва — 2003

АННОТАЦИЯ

регистрационный номер

К работе «Система автоматического оповещения пользователей в локальной сети»

Автор: **Давыдова Мария Владимировна**
Лицей №1580, 11 класс

Научный руководитель: **Дединский Илья Рудольфович**
преподаватель программирования лицея №1580

Разработан программный комплекс типа клиент-сервер, автоматизирующий процессы отправки и приема текстовых сообщений, содержащих информацию для пользователей, на все рабочие станции сети. Клиентская часть работает под ОС DOS в модифицированном текстовом режиме как резидентная программа (согласно техническому заданию) и демонстрирует принимаемые сообщения на 26 строке экрана. Для плавного появления сообщения используется динамическое перепрограммирование знакогенератора. Также разработаны: открытый протокол взаимодействия между компонентами комплекса, надстройка над сетевым протоколом Novell IPX, реализующая асинхронную работу с пакетами данных, и компоненты, обеспечивающие рассылку и отображение текста сообщения на рабочей станции. Комплекс оставляет возможность изменения отдельных его компонентов, а также добавления новых, для решения других задач. Область применения комплекса: компьютерные классы, небольшие сети с невысокой пропускной способностью. Язык разработки — Turbo Assembler 4.1 (резидентная часть клиента), Borland C++ 3.1 (инсталляционная часть клиента, сервер).

СОДЕРЖАНИЕ

Список условных обозначений, сокращений и терминов

BIOS (Base Input Output System) – базовая система ввода-вывода

CRTC (CRT Controller) – микроконтроллерный блок управления электронно-лучевой трубкой в EGA/VGA

EGA/VGA (Enhanced Graphics Adapter/Video Graphics Array) – распространенный стандарт видеоадаптеров, на который ориентирована настоящая разработка

ECB (Event Control Block) – блок с информацией для драйвера IPX

ESR (Event Service Routine) – обработчик события отправки/приема пакета в IPX

IPX (Advanced Netware Internetwork Packet Exchange Protocol) – сетевой протокол, использующийся в сетях Novell

ISR (Interrupt Service Routine) – обработчик прерывания

SAP (Service Advertising Packet) - пакет с сервисной информацией

Введение

В процессе эксплуатации локальных вычислительных сетей часто случается ситуация в которой администратору приходится информировать о чем-либо всех пользователей, работающих на всех рабочих станциях это сети. Часто эта информация не так важна, чтобы пользователь прерывал текущую работу. Во многих случаях достаточно, чтобы некоторое количество пользователей просто обратило внимание на сообщение. Это может быть, например, может быть информация о новом ресурсе сети и т.п.

Таким образом, возникает задача информирования «в фоновом режиме». Существует некоторое количество программ, с помощью которых можно отправлять сообщения, используя сеть. Такими программами являются, например, ICQ, утилита Novell Send и др. Однако, большинство программ такого рода предназначены для общения между пользователями. Отправка больших объемов однотипной информации всем пользователям сети с их помощью затруднительна. К тому же почти всякая из программ подобного рода предполагает какие-либо действия пользователя в ответ на пришедшее сообщение (нажатие клавиш и т.п.).

Для информирования «в фоновом режиме» неплохо подходит бегущая строка с периодически меняющимися сообщениями. Необходимо, чтобы бегущая строка не мешала работе ни пользователя, ни других программ, установленных на компьютере.

Сегодня существует один программный продукт, решающий схожую задачу – RIAN Ticker Tape. Эта программа выводит на экран бегущую строку с новостями РИАН, которые загружаются с сайта www.rian.ru. Программа разработана под операционную систему Windows. К сожалению, в наше время еще существуют компьютеры, конфигурации которых недостаточны для успешной работы Windows. Часто в таких случаях компьютер работает под управлением MS-DOS.

Анализ информации, представленной в интернете, не выявил схожие программные продукты, предназначенные для работы под MS-DOS. Это и послужило основание для разработки данного программного продукта.

Постановка задачи: разработка программного комплекса, предназначенного для автоматизации процессов отправки и приема текстовых сообщений, содержащих информа-

цию для пользователей, на рабочие станции сети. Комплекс должен удовлетворять следующим требованиям:

1. Работа под управлением MS-DOS и поддержкой протокола Novell IPX;
2. Работа на маломощных компьютерах и сетях с невысокой пропускной способностью;
3. Надежность и простота использования;
4. Психологическая ненавязчивость интерфейса.

Цели работы:

1. Разработка открытого протокола взаимодействия программ-компонентов комплекса (см. раздел 5);
2. Разработка программно-независимой надстройки над протоколом IPX, обеспечивающей асинхронную работу с пакетами данных, использующей механизм таймаутов (см. раздел 4.2);
3. Разработка программы-компонента реализующей механизм бегущей строки (см. раздел 2);
4. Разработка программы-компонента координирующей работу комплекса (см. раздел 3).

Область применения комплекса: комплекс может быть использован в компьютерных классах, небольших сетях, на рабочих станциях которых работает MS-DOS, для информирования пользователей.

1. Структурно-функциональное построение программного комплекса

Поставленная задача реализуется в виде комплекса из двух программ–компонентов. Первый компонент, MSGShow – принимает текст сообщения и выводит его на экран. Он устанавливается на каждой рабочей станции. Второй компонент, MSGSend – программа-менеджер сообщений, распределяет и посылает задачи для каждой копии MSGShow, работающей в сети. Он должен быть запущен на одном из компьютеров сети. В настоящее версии программа рассылки сообщений так же должна исполняться на рабочей станции под управлением MS-DOS. Однако существование протокола взаимодействия (см. раздел 5) между компонентами не делает это ограничение обязательным в будущих версиях. Так же открытость протокола позволяет беспрепятственно добавлять в комплекс новые программы-компоненты для решения других задач. Структурная схема программного комплекса представлена на рис. 1.1.

Приводимые принципы призваны упростить все процессы, связанные с разработкой и сопровождением комплекса:

1. Независимость компонентов комплекса;
2. Наличие единого протокола взаимодействия между компонентами;
3. Максимально возможное разделение функций компонентов;
4. Централизованное управление компонентами;

Ошибка! Объект не может быть создан из кодов полей редактирования.

Рис 1.1. Структурная схема программного комплекса

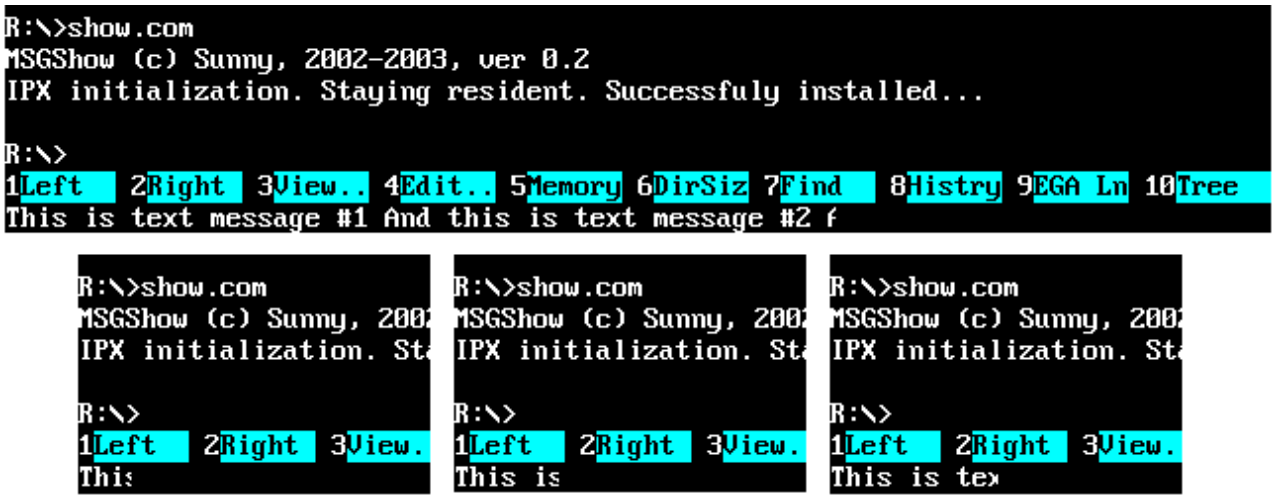
2. Компонент MSGShow

MSGShow – резидентная программа. Для вывода на экран дополнительной, двадцать шестой, текстовой строки, изменяются установки контроллера электронно-лучевой трубки (CRT Controller). Для достижения эффекта плавного появления надписи динамически переопределяется основной знакогенератор (см. рис 2.1).

Вся программа (за исключением инициализации IPX) написана на языке ассемблера. Ассемблер предпочтительней языков высокого уровня по двум причинам:

1. Критичность размера и времени выполнения кода
2. Удобства языка ассемблера для решения низкоуровневых задач

Для разработки использовались: трансляторы Turbo Assembler 4.1 и Borland C++ 3.1, компоновщик Turbo Link 4.0, отладчик Turbo Debugger 3.1.



(а) (б) (в)

Рис. 2.1. Внешний вид 26 строки на различных стадиях вывода символов.
а, б, в — последовательные стадии появления надписи.

2.1. Модули компонента

При разработке MSGShow был разделен на четыре независимых модуля:

1. Основной модуль (main.asm). Его функции – перехват необходимых прерываний, слежение за другими копиями программы в памяти, работа с памятью, корректное

завершение программы, вызов других модулей. Так же он обеспечивает поддержку клавиатуры и контроль видеорежима.

2. Модуль работы со знакогенератором (shift.asm). Динамически перепрограммирует знакогенератор, изменяя вид символов, напрямую работает с видеопамятью, выводя на экран переопределенные символы.
3. Модуль работы с видеоадаптером (26line.asm). Не использует прерывание 10h. Сохраняет изначальное состояние регистров CRTC, изменяет видеорежим и восстанавливает его, когда необходимо.
4. Модуль работы с сетевым протоколом IPX (ipxint.asm). Обеспечивает поддержку протокола взаимодействия программ-компонентов. Использует библиотеку ipxlib.asm (см. раздел 4.2).

Все модули реализованы в виде библиотек процедур, что делает возможным их дальнейшее независимое использование.

2.2. Модуль работы со знакогенератором

Строка с сообщением - это последовательность символов появляющихся на экране. За один шаг строка расширяется на величину от одного пикселя до одного знакоместа (9 пикселей). При заполнении всей строки (80 знакомест), она очищается, и следующий символ появляется в начале, в крайнем левом положении.

2.2.1. Принцип работы знакогенератора

Видеопамять расположена, начиная с сегмента 0b800h[1] смещения 0000h. Она разделена на несколько видеостраниц, расположенных последовательно друг за другом. Видеостраницы нумеруются последовательно, начиная с 0. Нулевая находится по адресу 0b800h:0000h, первая – 0b800h:4096h и так далее. Переключение между видеостраницами осуществляется средствами функции 05h прерывания 10h. Есть понятие текущей видеостраницы, все, что пишется в область памяти текущей видеостраницы, сразу отображается на экране. На любой видеостранице расположен образ экрана в виде ASCII-кодов символов и цветовых кодов, так называемых атрибутов. Строение видеобуфера показано на рис. 2.2.

[1] Все дальнейшие указания цифр или адресов памяти даны для стандартного текстового режима (#3 по стандартной таблице BIOS EGA/VGA – функция 0, прерывание 10h).

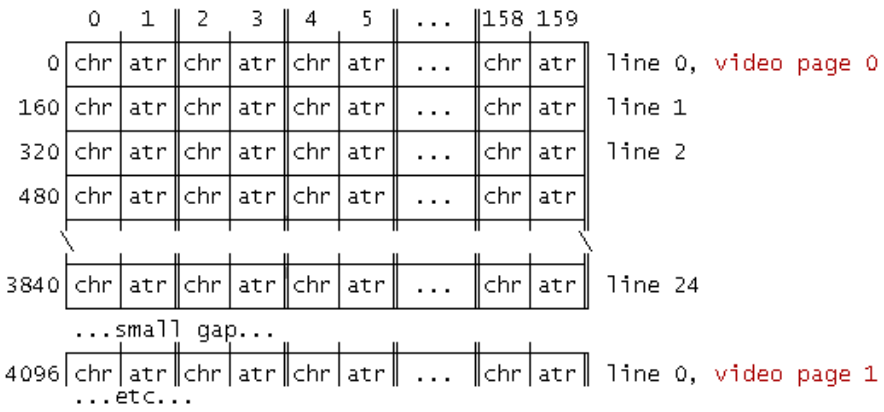


Рис.2.2. Схема строения видеобуфера [из TECH Help! 6.0]

Для хранения одного символа в видеопамяти используется два последовательных байта. Первый байт – ASCII-код символа. Второй (атрибут) содержит всю информация о том, как должен отображаться символ: его цвет, цвет фона, яркость, мигание, номер знакогенератора. Строение байта-атрибута показано на рис. 2.3.

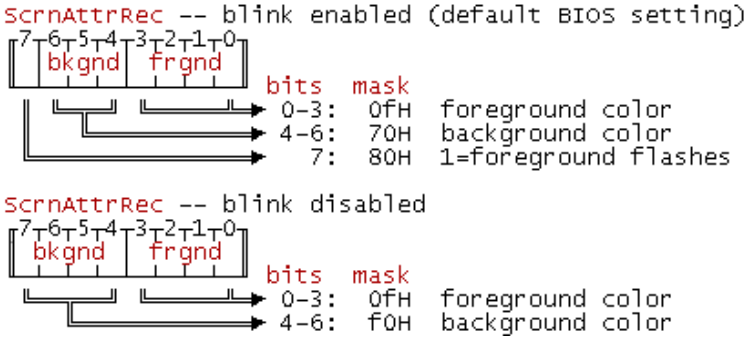


Рис. 2.3. Схема строения байта-атрибута [из TECH Help! 6.0]

Изображение каждого символа задается битовой матрицей 8*16. Для описания одного символа требуется 16 (не 128!) байт. Каждый байт определяет одну сканстроку изображения. Один стандартный набор (знакогенератор) включает в себя 256 символов. Таким образом, знакогенератор – это таблица из 256d*16d = 4096d байт[1]. Смещение для нужного символа вычисляется как 0ah*(ASCII-код символа). Одновременно в памяти может храниться от четырех (EGA) до восьми (VGA) знакогенераторов, но изображаться на экране может не более двух. Пример битовой матрицы одного символа приведен на рис. 2.4.

В базовой системе ввода-вывода (BIOS) существует функция для работы со знакогенераторами, доступная по прерыванию 10h под номером 11h.

Список основных подфункций, использованных в программе:

[1] Поскольку в тексте используются как шестнадцатеричные (h), так и десятичные (d) значения, то система счисления каждый раз уточняется.

1100h: Загрузка предварительно определенного шрифта. Для переопределения нужных символов.

1103h: Одновременное использование двух знакогенераторов. Номер используемого набора символов задает третий бит атрибута (в обычном режиме он отвечает за яркость).

1114h: Загрузка и активация стандартного шрифта 8*16. Для восстановления испорченного знакогенератора.

1130h: Получение информации о любом стандартном шрифте, включая высоту в сканстроках, а так же сегментный адрес описания стандартного шрифта.

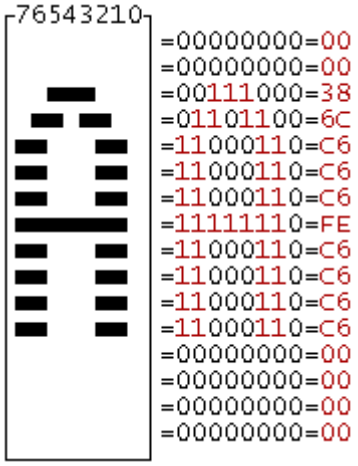


Рис. 2.4. Битовая матрица символа с ASCII кодом 65h [TECH Help! 6.0]

2.2.2. Динамическое переопределение знакогенератора

Для создания иллюзии плавного движения применяется следующая техника. Из знакогенератора выбирается наименее используемый символ, далее называемый буферным. Вид буферного символа постоянно переопределяется в составляющие полного изображения искомого символа. MSGShow в качестве буферного символа использует символ с ASCII-кодом 0ffh.

Последовательность появления одного символа с шагом 2 пикселя (см. рис. 2.5):

1. Обнуление матрицы 0ffh. Установка в текущее место строки 0ffh.
2. Загрузка описания матрицы символа. Ее модификация (видны два столбца). Переопределение 0ffh новой матрицей.
3. Загрузка матрицы нужного символа. Ее модификация (видны четыре столбца). Переопределение 0ffh новой матрицей.

4. Загрузка матрицы нужного символа. Ее модификация (видны шесть столбцов). Переопределение 0ffh новой матрицей.
5. Установка в текущее место строки реального символа (видны все восемь столбцов). Обнуление матрицы 0ffh.

2.3. Модуль работы с CRT-контроллером[1]

Для вывода информации программа MSGShow использует 26 строку в стандартном текстовом режиме #3. Стандартная высота экрана в этом режиме – 25 строк. Дополнительная строка создается искусственно с помощью перепрограммирования CRT контроллера. Все программы, рассчитанные на этот режим, используют только 25 строк. Поэтому работа на 26 строке не будет мешать ни одной из уже работающих программ.

Для понимания работы алгоритма создания 26 строки рассмотрим процесс формирования изображения видеоадаптером, на примере EGA/VGA, так как он поддерживается всеми современными адаптерами и MSGShow работает именно с ним.

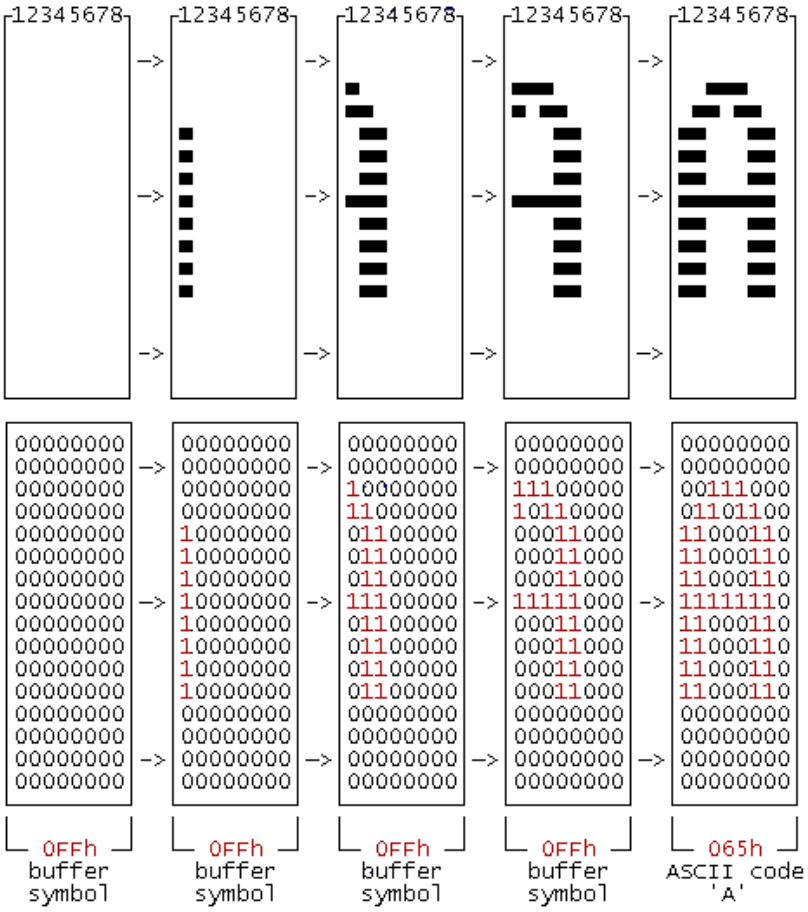


Рис. 2.5. Иллюстрация к алгоритму плавного появления символа на экране.

[1] В этом разделе не приводится исчерпывающее техническое описание работы видеоадаптреа, а лишь указываются отдельные моменты, принципиальные для работы программы.

2.3.1. Общая структура видеоадаптера EGA/VGA

Адаптер состоит из следующих компонентов:

1. Видеобуфер (Display Buffer). Он же память адаптера, или видеопамять.
2. Графический контроллер (Graphics Controller). Направляет информацию из видеобуфера в контроллер атрибутов и процессор. В графических режимах данные передаются последовательно, а в текстовых параллельным потоком, используя четыре битовые плоскости, что позволяет записывать 32 бита за одно обращение к памяти.
3. Контроллер атрибутов (Attribute Controller). Получает данные из видеобуфера и преобразует их в сигналы для монитора. Здесь же управляется палитра, мигание и подчеркивание.
4. Блок синхронизации (Sequencer). Здесь генерируются тактовые сигналы и сигналы доступа к видеопамяти. С помощью него обеспечивается доступ процессора к видеопамяти в короткие промежутки времени, когда видеобуфер не занят графическим контроллером во время формирования изображения.
5. Блок управления электронно-лучевой трубкой (CRT Controller, CRTC). Его функция – управление сигналами вертикальной и горизонтальной синхронизации, начальным адресом видеобуфера, положением и формой курсора и др.

Структурная схема адаптера приведена на рис. 2.6.

Все компоненты адаптера программно управляемы.

Стандартный сервис обслуживания входит в состав BIOS и доступен по прерыванию 10h. В BIOS есть несколько стандартных, заранее определенных, режимов работы адаптера. Переключение между ними возможно с помощью функции 0 прерывания 10h.

Каждый компонент адаптера имеет несколько регистров, используемых для управления его функциями. Изменения видеорежима есть изменение значений конкретных регистров. Таким образом, каждый заранее определенный режим – это набор значений для всех регистров видеоадаптера.

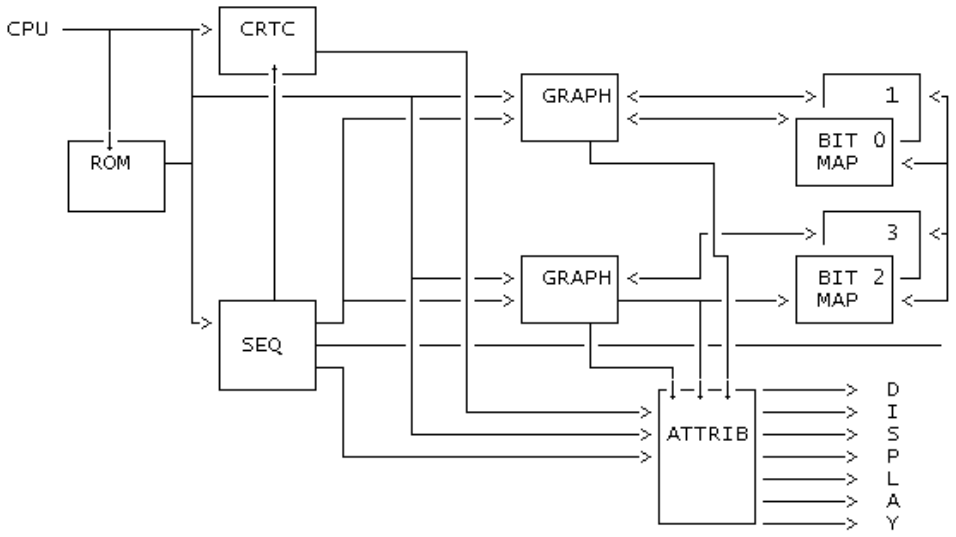


Рис. 2.6. Структурная схема видеоадаптера EGA/VGA. CPU – центральный процессор, ROM – постоянная память адаптера, CRTC – контроллер электронно-лучевой трубки, GRAPH – графический контроллер, BIT MAP N – одна из четырех битовых плоскостей, ATTRIB – контроллер атрибутов.

Итак, видеорежимы характеризуются следующими параметрами:

1. Вертикальное разрешение (количество строк растра на экране)
2. Горизонтальное разрешение (количество символов или пикселей на экране)
3. Представление данных в видеобуфере
4. Атрибуты вывода (цвет, мигание и т.д.)

Горизонтальное и вертикальное разрешения определяются согласованными по времени сигналами электронно-лучевой трубки. Луч постоянно движется по растру по строкам сверху вниз, успевая обойти весь экран 50-90 раз в секунду.

В процессе вывода каждой строки интенсивность луча изменяется в соответствии с сигналами, подаваемыми видеоадаптером. В случае цветного изображения отдельно задается интенсивность каждого из базовых цветов. Луч движется с постоянной скоростью, и всякий раз при прохождении строки растра до конца генерируется сигнал обратного горизонтального хода луча. Также при прохождении всех строк экрана генерируется сигнал обратного вертикального хода луча, после которого луч движется назад к началу первой строки.

Адаптер программируется так, чтобы время вывода данных из буфера в кадре было меньше времени вывода всего кадра, в таком случае вокруг изображения образуется бордюр.

1. Программирование непосредственно адаптера включает в себя:
2. Программирование блока управления электронно-лучевой трубкой;
3. Программирование блока синхронизации;

4. Задание частоты генератора пикселей;
5. Задание высоты символов в строках растра;
6. Модификация требуемых переменных BIOS;

2.3.2. Перепрограммирование адаптера для получения 26 строки в текстовом режиме

Для получения 26 строки надо изменить значения трех регистров CRT-контроллера, находясь в стандартном текстовом режиме:

1. Регистр начала вертикального гашения луча (Vertical Blanking Start Register). Содержит номер строки растра, после которой вызывается сигнал вертикального гашения луча. Требуется прибавить к нему текущую высоту символа.
2. Регистр длительности участка отображения в кадре (Vertical Display End Register). Содержит номер последней строки растра, отображаемой в кадре. Требуется прибавить к нему текущую высоту символа.
3. Регистр начала вертикального обратного хода луча (Vertical Retrace Start Register). Содержит номер строки, с которой начинается вертикальный обратный ход луча. Требуется прибавить к нему половину высоты символа, что бы центрировать экран.

Текущая высота символа в сканстроках храниться в регистре вертикального размера символа (Maximum Scan Lines Register). Младшие четыре бита этого регистра определяет количество сканстрок на одно знакоместо, оно должно быть равно количеству сканстрок минус один.

Особенность работы с регистрами CRT-контроллера заключается в том, что необходимо сначала узнать номер адресного регистра, через который происходит адресация к другим регистрам по номерам. Также требуется сохранять значения регистров до изменения и восстанавливать их всякий раз, как только какая-либо другая пытается изменить видеорежим.

2.4. Основной модуль

2.4.1. Анализ работы резидентных программ

Резидентная программа – это программа, остающаяся в памяти после возвращения управления DOS и получающая управление по прерываниям, перехваченным ею.

Перехваченное прерывание – это прерывание, у которого вместо стандартного обработчика (Interrupt Service Routine, ISR) вызывается какой-либо другой обработчик, обычно определенный перехватывающей программой.

Резидентной программе необходимо оставаться в памяти после завершения. Для этого используется функция 31h прерывания 21h (DOS). Перед выходом необходимо рассчитать, сколько параграфов (1 параграф = 16 байт) памяти занимает резидентная часть, начиная с PSP и заканчивая меткой в начале инсталляционной части (см. рис. 2.7, метка ResEnd). Функция 31h завершает программу, оставляя занятым блок памяти, указанного размера. Блок начинается с адреса cs:0000h. Структура блока памяти резидентной программы показана на рис. 2.7.

Перехват прерываний позволяет резидентной программе получать управление в результате нужного события. Сегментные адреса ISR всех прерываний по порядку хранятся в так называемой таблице прерываний, расположенной по адресу 0000h:0000h. Схема таблицы прерываний показана на рис. 2.8.

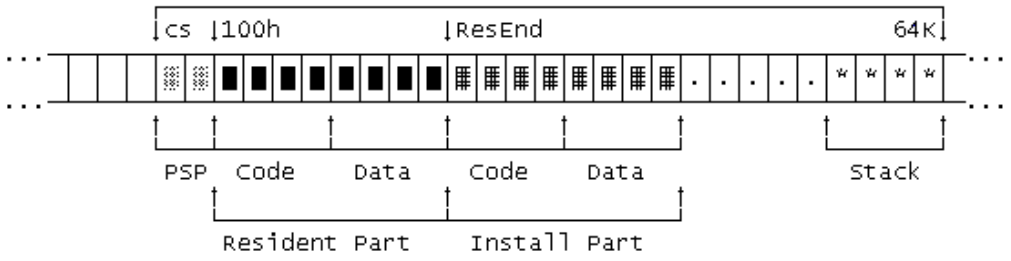


Рис. 2.7. Структура блока памяти резидентной программы.

Адрес обработчика N-ого прерывания расположен по адресу 0000:N*4. Вписав вместо него адрес своей процедуры-обработчика, программа перехватывает прерывание N. Перед перехватом нужно сохранить адрес предыдущего ISR и вызвать его в конце собственной процедуры обработки прерывания. В это случае говорят о цепочке обработчиков одного прерывания. Таким образом, все резиденты, ранее перехватившие это прерывание, тоже получают управление.

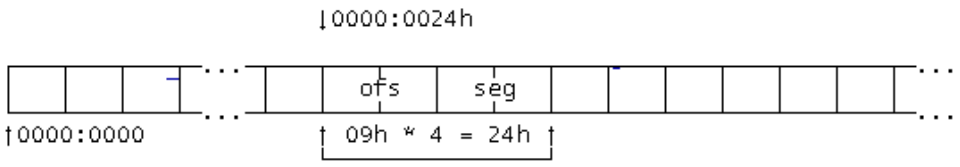


Рис. 2.8. Таблица прерываний и адреса ESR прерываний.

2.4.2. Прерывания 08h, 09h, 10h, 2fh, 28h и их обработчики

Для обеспечения корректной работы программы MSGShow требуется перехват следующих прерываний:

1. Аппаратное прерывание таймера (int 08h). Оно вызывается через каждые 55 мс, 18.2 раза в секунду. Именно его обработчик выполняет все действия, связанные с логикой работы программы, т.е. передвигает бегущую строчку и поддерживает связь с MSGSend.
2. Аппаратное прерывание клавиатуры (int 09h). Вызывается всякий раз, когда на клавиатуре нажата клавиша. Через него отслеживаются «горячие» клавиши программы. Поддержка клавиш Cntrl и Alt сделана путем частой (каждый вызов), проверки байта состояния клавиатуры (0040h:0017h).
3. Прерывание Video BIOS (int 10h). В нем перехватывается функция 00h (изменение видеорежима). Как только какая-либо программа пытается изменить видеорежим, необходимо временно отключить бегущую строку и вернуть значения регистров CRTC в первоначальное состояние, а по возвращении в стандартный текстовый режим, снова активизировать MSGShow. Этим и занимается обработчик прерывания 10h.
4. Мультиплексное прерывание (int 2fh). Это прерывание предоставляет каждой программе возможность записать себя в цепочку мультиплексных процессов. Каждой программе присваивается уникальный идентификатор (MULTIPLEX ID). Программа перехватывает прерывание 2fh и отслеживает вызов, содержащий свой идентификатор. В ответ не него она отсылает код, говорящий о том, что копия программы уже загружена в память. При каждом запуске MSGShow вызывает прерывание 2fh со своим идентификатором и, если программа уже загружена, она завершается. Так же обработчик этого прерывания исполняет функцию API программы, через вызов 2fh программе сообщаются опции заданные в командной строке.
5. Прерывание Dos Safe (int 28h). Основное прерывание MS-DOS (int 21h) не реентерабельно. Это значит, что если прерывание DOS один раз вызвано, нельзя вызывать его еще раз, пока предыдущий вызов не завершится. Если к функциям DOS обращается резидентная программа, ей не известно, вызывала ли какая-либо другая программа int 21h. Специально для резидентов существует прерывание 28h. Оно вызывается каждый раз, когда DOS свободен и обращение к его функциям безопасно. В MSGShow перехват этого прерывания обеспечивает корректное ос-

вобождение блока памяти после завершения программы, используя функцию 49h прерывания 21h.

2.5. Системные требования и методическое обеспечение

2.5.1. Системные требования

Минимальные системные требования:

Аппаратные требования: Процессор Intel 80386, видеоадаптер VGA/SVGA, сетевая карта, клавиатура.

Программные требования: Операционная система MS-DOS 3.30+, установленный драйвер протокола Novell IPX.

Рекомендуемые системные требования совпадают с минимальными.

2.5.1. Методическое обеспечение

1. Установка. Скопируйте файл msgshow.com на жесткий диск компьютера.
2. Запустите файл msgshow.com. Если инициализация прошла успешно, внизу экрана появится еще одна текстовая строка, на которой вскоре появятся текстовые сообщения.
3. Теперь можно увеличивать/уменьшать скорость появления сообщений комбинациями клавиш: Ctrl-Alt-Right/Ctrl-Alt-Left.
4. Программа поддерживает следующие опции командной строки:
   /? – помощь;
   /m – сообщить количество памяти, которое занимает MSGShow;
   /n – сообщить ID текущей копии MSGShow. Под этим номером программа известна компоненту MSGSender;
   /t – завершить работу программы, отключиться от MSGSend, восстановить установки CRT контроллера;
5. Если корректная работа по какой-либо причине не возможно, MSGShow не запуститься и сообщит об ошибке. При этом передача управления MS-DOS происходит с кодом выхода 01h. В этом случае следует устранить причину ошибки и запустить программу вновь.
   1. MSGShow ERROR 01: Program already loaded – программа уже запущена и сейчас работает.

2. MSGShow ERROR 02: Wrong video mode – неверный начальный видеорежим, программа может работать только в стандартном видеорежиме #3.

3. Компонент MSGSend

Программа MSGSend призвана координировать все копии программы MSGShow. MSGSend может быть запущена только на одном компьютере в сети и должна работать постоянно. Язык исполнения – Си++, для разработки использовался транслятор Borland C++ 3.1.

3.1. Принцип работы компонента

1. На вход MSGSend подается текстовый файл, содержащий информацию, посылаемую в сеть.
2. Файл анализируется и на его основе в оперативной памяти создается нумерованный список так называемых записей – структур, содержащих текст с указанием его длины в байтах.
3. MSGSend периодический посылает в сеть по широковещательному адресу (0FFFFFFFFFFFh) так называемый пакет с сервисной информацией (SAP, Service Advertising Packet), что бы любая из вновь подключившихся программ MSGShow могла инициализироваться. Этим же пользуется и MSGSend при попытке запустить еще одну копию программы в сети. Новая копия перестанет работать, как только получит пакет с сетевым адресом уже существующей MSGSend. Протокол взаимодействия компонентов описан в разделе 5.
4. Каждая новая копия MSGShow регистрируется в отдельном списке учетных записей. В состав учетной записи входит сетевой адрес станции, на которой выполняется данная копия MSGShow, номер текущей записи, выводимой на экрна в данный момент на данной станции.
5. При получении запроса на новый текст MSGSender проверяет, какая запись только что была отправлена этой копии MSGShow. Далее, исходя из определенных заранее условий, решает, какая следующая текстовая запись будет послана.
6. Если приходит запрос на уничтожение учетной записи (одна из копий MSGShow выгружается из памяти), то запись удаляется из памяти.

3.2. Формат входного файла

Информация для отправки на все рабочие станции храниться в текстовом файле в виде записей. Одна запись – одна строка, длина строки не ограничена, признаком ее конца считается символ «\n» (последовательность байтов 0dh, 0ah). Запись не может быть длиннее восьмидесяти символов. Если строка в файле длиннее, то она разбивается на несколько записей. На логике подачи информации это не отражается, так как такие записи подаются в сеть последовательно.

3.3. Структура классов компонента

Для реализации логики работы программы было написано несколько классов. Диаграмма наследования приведена на рис. 3.1.

1. CObj – базовый класс, из него наследуются все остальные классы программы. Имеет две чисто виртуальные функции: isOk() (для определения текущего состояния объекта) и dump() (для отображения текущего состояния объекта в файл или на экран), а так же переопределенные операторы сравнения и виртуальную функцию compare() (для сравнения объектов).
2. CList – динамический двунаправленный список. Т.к. данные программы однородны по типу, но не постоянны по количеству и последовательности, то такой список – лучшее решение для их хранения. При добавлении нового элемента к списку указывается номер, по которому и происходит дальнейшая идентификация элемента. Этот номер используется как ID для MSGShower.
3. CNode и CRecord – классы, описывающие основные понятия, с которыми работает программа. CNode – учетная запись рабочей станции на которой работает MSGShow. CRecord – запись, часть информации, подаваемой в сеть с указанием ее длины.
4. CNet и CInfo – классы-оболочки соответственно для CList<CNode> и CList<CRecord>. Призваны ограничивать свободу обращения к таким спискам, что бы избежать порчи данных.
5. CDump[1] – вспомогательный класс, написанный для удобства отладки программы. Записывает в файл или выводит на экран текущее состояние нужных объектов. Основан на том, что в каждом классе есть виртуальная функция dump(), унаследованная из CObj.

[1] Дамп (англ. dump) – отладочная печать.

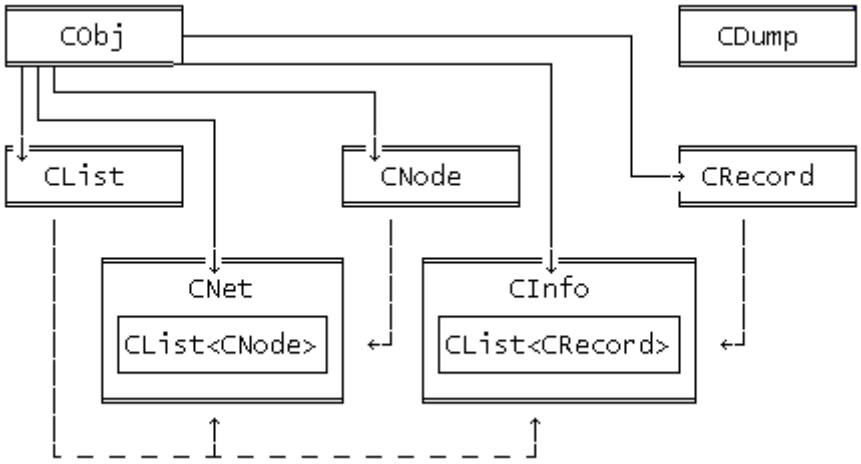


Рис. .3.1. Диаграмма наследования классов программы MSGSend.
Сплошная линия указывает наследование, пунктирная – включение.

3.4. Системные требования и методическое обеспечение

3.4.1. Системные требования

Минимальные системные требования:

Аппаратные требования: процессор Intel 80386, сетевая карта, клавиатура.

Программные требования: операционная система MS-DOS 3.30+, установленный драйвер протокола Novell IPX.

Рекомендуемые системные требования совпадают с минимальными.

3.4.2. Методическое обеспечение

1. Установка. Скопируйте файл msgsend.exe на жесткий диск компьютера.
2. MSGSend запускается с параметрами, задаваемыми с помощью командной строки.
   Syntax: MSGSEND infofile [dump options] [dumpfile]
   Первым и обязательным параметром является имя файла, который будет передаваться всем копиям MSGShow. Дальнейшие, не обязательные, параметры определяют параметры дампа. Сначала перечисляются все разрешенные уровни дампа. После них указывается имя файла, в который дамп будет записываться. Если имя файла-дампа не указано, то дамп будет производиться на экран. Если параметры дампа не указаны, то дамп производиться не будет.
3. Допустимые уровни дампа:

/1 – краткий информационный дамп, отображаются только события, связанные с логикой работы комплекса

/2 – дамп пакетов, отображаются приходящие и отправляемые пакеты в шестнадцатеричном формате с указанием их кодов (см. раздел 5)

/3 – дамп функций IPX, отображаются все вызовы стандартных функция драйвера протокола IPX

/4 – дамп состояния протокола, отображаются все изменения состояния протокола, такие как вызов ESR, таймауты, изменение флагов ит.п.

/5 – отладочный дамп, отображение прочей информации о работе MSGSend

4. Формат записи дампа:

   dumplevel yymmddhhmmss: componentname: comment: dumpinfo

   dumplevel – цифровое обозначение уровня дампа, к которому относиться данная запись

   yymmddhhmmss – дата и время записи

   componentname – имя компонента, сделавшего запись

   comment – комментарий по смыслу записи, обычно, название уровня дампа

   dumpinfo – непосредственно информация

5. Если корректная работа по какой-либо причине не возможна, MSGSend не запуститься и сообщит об ошибке. При этом передача управления MS-DOS происходит с кодом выхода 01h. В этом случае следует устранить причину ошибки и запустить программу вновь.
   1. MSGSend ERROR 01: Cannot open info file – невозможно открыть или создать файл с информацие для отправки в сеть
   2. MSGSend ERROR 02: Cannot open info file – невозможно открыть или создать файл для записи дампа
   3. MSGSend ERROR 03: No info file – файл с информацией для отправки в сеть не был указан при запуске программы
   4. MSGSend ERROR 04: MSGSend is already working in the net – одна копия MSGSend уже запущена и успешно работает в этой сети

3.4.3. Пример отладочной печати

Приведен дамп MSGSend, запущенного с опциями командной строки:

```
MSGSEND info.txt /1 /2 dump.txt
```

```
//MSGSend dump file dump.txt begin at Wed Mar 19 22:51:22 2003

1 030319105128: MSGSend: Broadcast [FFFFFFFFFFFF]: Service advertising
2 030319105129: MSGSend: Node ??   [0003FFFDFFFF]: Packet code 01
                packetR: 01 00 03 FF FD FF FF 00 00 00 00 00 00 00 00
                packetS: 02 01 00 00 00 00 00 00 00 00 00 00 00 00 00
1 030319105129: MSGSend: Node ??   [0003FFFDFFFF]: Connected as Node 01
2 030319105131: MSGSend: Node 01   [0003FFFDFFFF]: Packet code 03
                packetR: 03 01 00 00 00 00 00 00 00 00 00 00 00 00 00
                packetS: 01 18 54 68 69 73 20 69 73 20 74 65 78 74 20
1 030319105131: MSGSend: Node 01   [0003FFFDFFFF]: Message sent, length
24
                         “This is text message...”
1 030319105134: MSGSend: Broadcast [FFFFFFFFFFFF]: Service advertising
2 030319105135: MSGSend: Node ??   [0003FFFEFFFF]: Packet code 01
                packetR: 01 00 03 FF FE FF FF 00 00 00 00 00 00 00 00
                packetS: 02 02 00 00 00 00 00 00 00 00 00 00 00 00 00
1 030319105135: MSGSend: Node ??   [0003FFFEFFFF]: Connected as Node 02
2 030319105137: MSGSend: Node 02   [0003FFFEFFFF]: Packet code 03
                packetR: 03 02 00 00 00 00 00 00 00 00 00 00 00 00 00
                packetS: 01 18 54 68 69 73 20 69 73 20 74 65 78 74 20
1 030319105137: MSGSend: Node 02   [0003FFFEFFFF]: Message sent, length
24
                         “This is text message...”
1 030319105140: MSGSend: Broadcast [FFFFFFFFFFFF]: Service advertising
2 030319105142: MSGSend: Node 01   [0003FFFDFFFF]: Packet code 03
                packetR: 03 01 00 00 00 00 00 00 00 00 00 00 00 00 00
                packetS: 01 1C 41 6E 64 20 74 68 69 73 20 69 73 20 74
1 030319105142: MSGSend: Node 01   [0003FFFDFFFF]: Message sent, length
28
                         “And this is another text message...”
1 030319105144: MSGSend: Broadcast [FFFFFFFFFFFF]: Service advertising
2 030319105148: MSGSend: Node ??   [010000000000]: Packet code 02
                packetR: 02 01 00 00 00 00 00 00 00 00 00 00 00 00 00
1 030319105148: MSGSend: Node 01   [0003FFFDFFFF]: Disconnected

//MSGSend dump file end at Wed Mar 19 22:52:05 2003
```

4. Сетевой протокол Novell IPX

Сетевой протокол IPX (Advanced Netware Internetwork Packet Exchange Protocol) – инструмент представленный фирмой Xerox. Он позволяет программам, работающим на рабочих станциях сети Novell Netware, связываться между собой, с серверами и другими устройствами сети.

Связь осуществляется по средствам сетевых пакетов, структурированных в соответствии с XNS (Xerox Network Standard). Каждая рабочая станция имеет свой уникальный сетевой адрес, что позволяет посылать индивидуальные пакеты. Возможен так же обмен пакетов между рабочими станциями находящимися в физически разных сетях. Процесс этот прозрачен и прост, так как IPX поддерживает сетевые мосты Netware.

4.1. Прием и отправка пакетов IPX

Обычно пакет IPX состоит из двух частей: заголовка и собственно данных (хотя возможно делить пакет и на большее количество частей). Заголовок пакета всегда занимает 30 байт, а весь пакет может быть длиной от 30 до 512 байт.

Информация о том, с каким пакетом работать в данный момент, сообщается драйверу IPX с помощью так называемых блоков управления событием (Event Control Block, ECB). В них указывается адрес заголовка пакета и его данных, а так же другая вспомогательная информация. Заполненный ECB можно поставить в очередь на отправку/прием.

Структура заголовка пакета представлена в Таблице 4.1, а структура ECB – в Таблице 4.2. (данные таблиц указываются в соответствии с IPX and AES Specifications of March, 1986 (c) Novell Inc.)

Таблица 4.1. Структура заголовка пакета IPX

Смещение	**Название поля**	**Размер**	**Описание**
0	Checksum	2 bytes	Контрольная сумма пакета
2	Length	2 bytes	Длина пакета вместе с заголовком
4	Transport Control	1 byte	Счетчик мостов
5	Packet Type	1 byte	Тип пакета
6	Destination Network	4 bytes	Номер сети получателя
10	Destination Node	6 bytes	Номер станции получателя
16	Destination Socket	2 bytes	Сокет программы получателя
18	Source Network	4 bytes	Номер сети отправителя

Смещение	Название поля	Размер	Описание
22 28	Source Node Source Socket	6 bytes 2 bytes	Номер станции отправителя Номер станции отправителя

Таблица 4.2. Структура ECB

Смещение	Название поля	Размер	Описание
0	Link	4 bytes	Адрес следующего в очереди ECB
4	ESR Address	4 bytes	Адрес обработчика ESR
8	In Use	1 byte	Флаг состояния
9	Completion Code	1 byte	Код завершения запроса
10	Socket Number	2 bytes	Номер сокета для отправки/приема
12	IPX Workspace	4 bytes	Рабочий буфер IPX
16	Driver Workspace	12 bytes	Рабочий буфер драйвера
28	Immediate Address	6 bytes	Адрес для передачи пакета
34	Fragment Count	2 bytes	Число фрагментов пакета
36	Fragment Descriptor 1	6 bytes	Фрагмент 1: размер, адрес
30+n*6	Fragment Descriptor n	6 bytes	Фрагмент n: размер, адрес

Подготовленные ECB и пакеты занимают место в очереди на отправку или прием. IPX не дает подтверждений после действий с пакетом. Их должна делать сама принимающая программа. Так же имеется возможность указать для каждого пакета свой ESR (Event Service Routine – обработчик события), которому будет передаваться управление после завершения действий с ECB блоком. Что бы послать/принять сетевой пакет нужно:

1. Инициализировать драйвер IPX
2. Подготовить различные блоки: пакет, заголовок и ECB.
3. Вызвать соответствующую функцию драйвера и передать в нее адрес ECB
4. Периодически проверять In Use флаг, находящийся в ECB. Пока он равен 1 – идет работа с пакетом, как только он сбрасывается в 0 – операция с пакетом завершена.

4.2. Реализация

Обе программы-компонента имеют в своем составе модули, поддерживающие работу с протоколом IPX. Каждая из программ содержит два заголовка пакета и ECB блока к ним: один пакет используется дли приема данных, другой – для отправки.

Использование ESR-обработчиков позволяет реализовать асинхронную работу с пакетами. То есть ECB отправляется в очередь и дальнейшая работа с ним не отслеживается, пока не вызывается ESR. Работа с сетью идет циклами: отправка-прием. Пока цикл не завершен, программы прерывают свою работу. Завершает цикл сообщение ESR (MSGSend)

или сброс соответствующего флага (MSGShow). Описание протокола взаимодействия см. в главе 5.

В циклах отправка-прием иногда случаются сбои по причине загруженности сети или слишком большом объеме приходящих пакетов. Так как IPX не осуществляет автоматического подтверждения приема пакета, считается, что пакет был доставлен успешно, если в течение определенного времени на него пришел осмысленный ответ. Если по истечении этого времени ожидаемого ответа нет, происходит событие «таймаут», то есть полный сброс текущих заданий и повторная посылка того пакета, на который не пришел ответ. Такая схема позволяет минимизировать потери информации при отправке с одной рабочей станции на другую.

Для удобства доступа к функциям протокола IPX написаны две библиотеки: ipx.cpp и ipxlib.asm, соответственно на Си++ и ассемблере. Первая используется программой MSGSend, вторая – MSGShow. Библиотеки предоставляют удобную оболочку инициализации IPX, вызовам стандартных функций и обработке ошибок.

Любые действия, связанные с протоколом, осуществляются только соответствующими модулями компонентов, поэтому переписывание комплекса под другой протокол, не составляет особенного труда.

5. Протокол взаимодействия компонентов

5.1. Описание протокола

В один момент времени в сети может работать только одна копия программы MSGSend и любое количество программ MSGShow.

1. При инициализации MSGShow принимает диагностическое сообщение с сетевым адресом станции, на которой работает MSGSend (см. рис. 5.1. сообщение (1)).
2. После этого MSGShow посылает сообщение с просьбой зарегистрировать себя (см. рис. 5.1. сообщение (2)).
3. Если регистрация проходит нормально, он получает назад пакет с индивидуальным номером, по которому и происходит дальнейшая идентификация этой копии программы и ведется статистика (см. рис. 5.1. сообщение (3)).
4. Дальнейшее общение происходит по следующей схеме: как только у MSGSend подходит к концу текущее сообщение, посылается запрос на следующее (см. рис. 5.1. сообщение (4)).
5. MSGSend, ведя статистику каждой станции, посылает новую информацию (см. рис. 5.1. сообщение (5)).
6. Если работа MSGShow завершена, то посылается сообщение с просьбой отключить данную копию от MSGSend и удалить учетную запись (см. рис. 5.1. сообщение (6)).

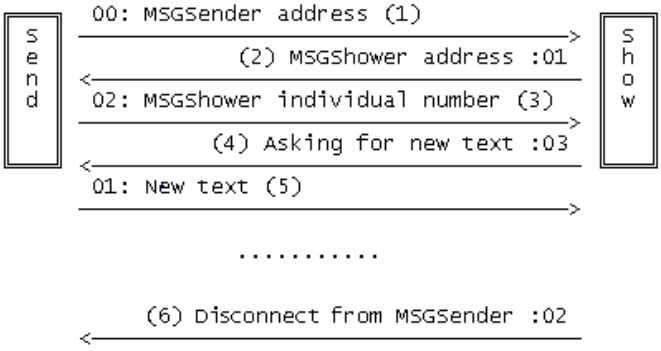


Рис. 5.1. Схема протокола взаимодействия компонентов
(номера в скобках соответствуют пунктам описания протокола)

Программа-компонент MSGShow лишена возможности знать, какое следующее сообщение придет, и сколько еще копий программы работает в сети в данный момент. Так реализуется принципы единственности управления и разделения функций программ-компонентов.

5.2. Формат пакета

Размер посылаемого буфера – 83 байта. 80 байт – максимальная длина сообщения, 1 байт – код операции, 2 байта – дополнительная информация.

Независимо от отправителя пакета, первый байт в нем – код операции. Значения кодов приведены в таблицах 5.1 и 5.2.

Таблица 5.1. Коды операций, приходящих от MSGShow

Код	**Описание**
03	Запрос на следующее сообщение для показа
01	Запрос на инициализацию
02	Отключение от MSGSend, удаление учетной записи

Таблица 5.2. Коды операций приходящих от MSGSend

Код	**Описание**
00	Адрес станции, на которой работает MSGSend
01	Следующее сообщение
02	ID новой копии MSGShow

Форматы пакетов, в зависимости от кода операции и отправителя приведены в таблицах 5.3. – 5.8.

Таблица 5.3. Пакет с кодом 03 от MSGShow

Смещение	**Размер**	**Описание**
00	1 byte	Код операции – запрос на следующее сообщение
01	2 bytes	ID данной копии MSGShow

Таблица 5.4. Пакет с кодом 01 от MSGShow

Смещение	**Размер**	**Описание**
00	1 byte	Код операции – запрос на инициализацию
01	6 bytes	Сетевой адрес рабочей станции

Таблица 5.5. Пакет с кодом 02 от MSGShow

Смещение	**Размер**	**Описание**
00	1 byte	Код операции – отключение от MSGSend

Смещение	Размер	Описание
01	2 bytes	ID данной копии MSGShow

Таблица 5.6. Пакет с кодом 00 от MSGSend

Смещение	Размер	Описание
00	1 byte	Код операции – сетевой адрес станции с MSGSend
01	6 bytes	Сетевой адрес рабочей станции

Таблица 5.7. Пакет с кодом 01 от MSGSend

Смещение	Размер	Описание
00	1 byte	Код операции – следующее сообщение
01	2 bytes	Длина сообщения в байтах
03	n bytes	Текст сообщения

Таблица 5.8. Пакет с кодом 02 от MSGSend

Смещение	Размер	Описание
00	1 byte	Код операции – ID данной копии MSGShow
01	2 bytes	ID новой копии MSGShow

6. Выводы

Таким образом, в ходе работы был разработан комплекс, успешно решающий поставленные перед ним задачи и оставляющий возможность улучшения, как любой программы-компонента, так и любого модуля программ. Также разработан открытый протокол связи между компонентами, надстройка над протоколом IPX и система безопасности, что позволяет разрабатывать и добавлять свои программы-компоненты для решения других задач.

Направления дальнейших разработок:

1. Разработка дружественного пользовательского интерфейса для программы MSGSend
2. Разработка новых форматов входных данных для MSGSend
3. Разработка и поддержка Messaging API
4. Разработка версии MSGSend под ОС Novell и установка его на сервере Novell Netware
5. Разработка программы-компонента и версии протокола для обмена личными сообщениями между пользователями.
6. Поддержка других сетевых протоколов, прежде всего TCP/IP.
7. Разработка программы-модуля для реализации бегущей строки под ОС Windows.

Список использованных литературных источников

1. Internetwork Packet Exchange Protocol (IPX) with Asynchronous Event Scheduler (AES). Specifications as of March 19, 1986 (c) 1986 by Novell Inc.
2. А. Фролов, Г. Фролов. Локальные сети персональных компьютеров. Использование протоколов IPX, SPX и NETBIOS. - М.: Диалог-МИФИ, т. 4, 1993, 160 стр.
3. R. Wilton. Programmer's Guide to PC and PS/2 Video Programming. – Washington: Microsoft Press, 1987.
4. EGA/VGA Руководство программиста. - 273 стр.
5. D. Rollins. TECH Help! Database version 6.0. Flambeaux Software, 1998.
6. В.Ю. Пирогов. Assembler. Учебный курс. - М.: Нолидж, 2001, 848 стр.
7. С. Прата. Язык программирования С++. Лекции и упражнения. Учебник. Пер.с англ. - К.: «ДиаСофт», 2001, 656 стр.
8. Б. Страуструп. Язык программирования С++. Третье издание. – М.: БИНОМ, 1993, 1098 стр.

ПРИЛОЖЕНИЕ
Исходные тексты программ

Файл MsgShow\MACRO.ASM

```
;-------------------------------------------------------------
;macro.asm, MSGShow ver 0.2, (c) Sunny, 2002-2003
;-------------------------------------------------------------

.pushs          macro
                push es ds si di bp
                endm

.pops           macro
                pop bp di si ds es
                endm

.putsmb         macro   smb
                push ax
                mov al, smb
                int 29h
                pop ax
                endm

.printEPmacro
                push ax
                mov al, byte ptr cs: entry_point
                int 29h
                mov al, byte ptr cs: entry_point + 1
                int 29h
                mov al, byte ptr cs: entry_point + 2
                int 29h
                mov al, byte ptr cs: entry_point + 3
                int 29h
                pop ax
                endm

.dumpmemmacro smth, len

                push ax bx cx
                xor cx, cx
dumpmemloop:    mov bx, offset smth
                add bx, cx
                mov al, byte ptr [bx]
                sub al, 30d             ;!!!!!!
                int 29h
                inc cx
                cmp cx, len
                jne dumpmemloop
                pop cx bx ax

                endm
;-------------------------------------------------------------
```

Файл MsgShow\CONST.ASM

```
;-------------------------------------------------------------
;const.asm, MSGShow ver 0.2, (c) Sunny, 2002-2003
;-------------------------------------------------------------

vmem            = 0b800h
dlength         = 80d
line_num= 25d
buff_smb= 0ffh
color           = 07h
;shift_num      = 2d                            ;1, 2, 4, 8
```

```
delay           = 0
left            = 4bh
right           = 4dh
esc             = 01h
debug_key       = 41h

socket          = 0420h
ipx_packet_type = 4
ipx_timeout     = 30d
packet_len      = 83d
;-------------------------------------------------------------
```

Файл MsgShow\DATA.ASM

```
;-------------------------------------------------------------
;data.asm, MSGShow ver 0.2, (c) Sunny, 2002-2003
;-------------------------------------------------------------

                db '<Text to display>'
;text           db '[This is test version]'
;text_len       db $ - offset text
text            db packet_len - 3 dup (?)
text_lendw 0

                db '<Current smb>'
text_curr       dw 0                    ;current reading pos in text
                db '<Scr line>'
line            db dlength dup (0)      ;copy of display line
end_flagdb 0                            ;end of text flag

                db '<Current symbol>'
smb             db 0
definition      db 16 dup (0)
def_with_shift  db 16 dup (0)
bit_curr        db 0                    ;current number of bits to display
                db '<Place>'
place           dw 0                    ;number in the line on the scrren

                db '<CRTC Data>'
CRTCPortdw ?
CRTCRegslabel byte
CRTCReg15       db ?
CRTCReg12       db ?
CRTCReg10       db ?
CRTCReg09       db ?

                db '<Options>'
mux_id          db 0e1h
shift_num       db 01
shift_num_new   db 01

                db '<Net data>'
sender_adr      db 01h, 01h, 01h, 01h, 01h, 01h;6 dup (?)
my_number       dw ?
;-------------------------------------------------------------
```

Файл MsgShow\MAIN.ASM

```
;-------------------------------------------------------------
;main.asm, MSGShow ver 0.2, (c) Sunny, 2002-2003
;-------------------------------------------------------------

.model tiny, C
locals          @@

;-------------------------------------------------------------
include         const.asm
include         macro.asm
;-------------------------------------------------------------

public          mux_init
public          resident_init
public          text
public          text_len
public          my_number
public          sender_adr
```

```
.code

start:          jmp res_end

;-------------------------------------------------------------
new08           proc
                push bx

                inc byte ptr cs: ipx_count

@@next:         cmp byte ptr cs:ipx_work, 01
                jne @@check_delay
                cmp byte ptr cs: ipx_count, ipx_timeout
                jbe end08

                .386
                .pushs
                pusha
                mov byte ptr cs: ipx_count, 00
                call cancel_all
                call set_esr
                call ask_new_text
                call s_fill_blocks
                popa
                .pops
                jmp end08

@@check_delay:  cmp word ptr cs:ticks, delay
                je @@do
                inc byte ptr cs:ticks
                jmp end08

@@do:           .386
                pusha
                .pushs
                push cs
                pop es
                push cs
                pop ds
                call do
                .pops
                popa

end08:          pop bx

                cmp byte ptr cs:remove_flag, 01
                jne no08
                call remove

no08:           pushf
                db 09ah                 ;call far
oldisr08dd 0

                iret
ticks           dw 0000h
remove_flag     db 0
ipx_workdb 0
ipx_count       db 0
                endp

;-------------------------------------------------------------
new09           proc
                push ax
                in al, 60h
                call kbd
                pop ax

no09:           pushf
                db 09ah                 ;call far
oldisr09dd 0

                iret
                endp

;-------------------------------------------------------------
new10           proc

                cmp byte ptr cs: in10_flag, 00
```

```
                jne @@call

                inc byte ptr cs: in10_flag
                cmp ah, 00
                jne @@call
                cmp al, 02
                je @@restore
                cmp al, 03
                jne @@call

@@restore:      mov byte ptr cs:restore_flag, 01

@@call:         pushf
                db 09ah                 ;call far
oldisr10dd 0

                cmp byte ptr cs:in10_flag, 00
                jne @@done

                cmp byte ptr cs:restore_flag, 01
                jne @@done

                .pushs
                pusha
                push cs
                pop ds
                push cs
                pop ds
                call disable_26line
                call enable_26line
                popa
                .pops

                mov byte ptr cs:in10_flag, 0
                mov byte ptr cs:restore_flag, 0

@@done:         iret
restore_flag    db 0
in10_flag       db 0
                endp

;---------------------------------------------------------
new28           proc

                cmp byte ptr cs: freemem, 01
                jne no28

                push ax bx es ds
                cli
                mov bx, cs                      ;free memory
                mov es, bx
                mov ah, 49h
                int 21h

                xor ax, ax                      ;restore vectors
                mov ds, ax

                mov bx, 28h*4
                mov ax, word ptr cs:oldisr28
                mov [bx], ax
                mov ax, word ptr cs:oldisr28+2
                mov [bx+2], ax
                sti
                pop ds es bx ax

no28:           pushf
                db 09ah                         ;call far
oldisr28dd 0

                iret
freemem         db 0
                endp

;---------------------------------------------------------
new2f           proc

                cmp ah, cs: mux_id
                jne @@not_me
                mov bh, mux_id
```

```
@@m:            cmp al, 'm'
                jne @@n
                push ds ax dx
                push cs
                pop ds
                mov ah, 09h
                mov dx, offset msg_m
                int 21h
                pop dx ax ds
                jmp @@me

@@n:            cmp al, 'n'
                jne @@t
                push ds ax dx
                push cs
                pop ds
                mov ah, 09h
                mov dx, offset msg_n
                int 21h
                pop dx ax ds
                jmp @@me


@@t:            cmp al, 't'
                jne @@?
                mov byte ptr cs: remove_flag, 01
                jmp @@me

@@?:            cmp al, '?'
                jne @@me
                push ds ax dx
                push cs
                pop ds
                mov ah, 09h
                mov dx, offset msg_?
                int 21h
                pop dx ax ds
                jmp @@me

no2f:
@@not_me:       db 0eah                 ;jump far
oldisr2f        dd 0

@@me:           iret
msg_m           db 'MSGShow uses '
memory          db 4 dup ('0')
                db 'h bytes', 0dh, 0ah, '$'

msg_n           db 'ID number of this MSGShow is '
id_num          db 4 dup ('0')
                db 'h', 0dh, 0ah, '$'


msg_?           db 'Hotkeys:', 0dh, 0ah
                db '   slower: ctrl-alt-left  ', 0dh, 0ah
                db '   faster: ctrl-alt-right ', 0dh, 0ah
               ;db '   remove: ctrl-alt-esc   ', 0dh, 0ah
                db 'Options:', 0dh, 0ah
                db '   /? - this help note', 0dh, 0ah
                db '   /m - number of using memory', 0dh, 0ah
                db '   /n - ID of this MSGShow', 0dh, 0ah
                db '   /t - terminate MSHShow', 0dh, 0ah
                db '$'

                endp

;---------------------------------------------------------
include         ipxlib.asm
include         ipxint.asm
include         shift.asm
include         26line.asm
include         data.asm

;---------------------------------------------------------
kbd             proc

;entry: al - scancode
                push es bx
```

```
                mov bx, 0040h
                mov es, bx
                mov bx, 0017h
                test byte ptr es:[bx], 00000100b;Ctrl
                je @@done
                test byte ptr es:[bx], 00001000b;Alt
                je @@done

                cmp al, left
                jne @@mb_right
                cmp byte ptr cs: shift_num, 1
                je @@done
                cmp byte ptr cs: shift_num, 8
                jne @@left4
                mov byte ptr cs: shift_num_new, 4
                jmp @@done
@@left4:        cmp byte ptr cs: shift_num, 4
                jne @@left2
                mov byte ptr cs: shift_num_new, 2
                jmp @@done
@@left2:        mov byte ptr cs: shift_num_new, 1
                jmp @@done

@@mb_right:     cmp al, right ;debug_key
                jne @@mb_esc
                cmp byte ptr cs: shift_num, 8
                je @@done
                cmp byte ptr cs: shift_num, 1
                jne @@rigth2
                mov byte ptr cs: shift_num_new, 2
                jmp @@done
@@rigth2:       cmp byte ptr cs: shift_num, 2
                jne @@rigth8
                mov byte ptr cs: shift_num_new, 4
                jmp @@done
@@rigth8:       mov byte ptr cs: shift_num_new, 8
                jmp @@done

@@mb_esc:       cmp al, esc
                jne @@done

                ;call remove
                mov byte ptr cs: remove_flag, 00

@@done:         pop bx es
                ret
                endp

;-------------------------------------------------------------
remove          proc

;Removes string from memory

                .pushs
                pusha

                call disconnect
                call cancel_all

                xor ax, ax
                mov ds, ax

                mov bx, word ptr ds:08h*4
                cmp bx, offset new08
                jne @@error

                mov bx, word ptr ds:09h*4
                cmp bx, offset new09
                jne @@error

                mov bx, word ptr ds:10h*4
                cmp bx, offset new10
                jne @@error

                mov bx, word ptr ds:2fh*4
                cmp bx, offset new2f
                jne @@error
```

```
                mov bx, word ptr ds:28h*4
                cmp bx, offset new28
                jne @@error

                push cs
                pop ds
                call disable_26line             ;kill 26 scr line

                ;ea - jmp far

                xor ax, ax                      ;restore vectors
                mov ds, ax

                cli
                mov bx, 08h*4
                mov ax, word ptr cs:oldisr08
                mov [bx], ax
                mov ax, word ptr cs:oldisr08+2
                mov [bx+2], ax

                mov bx, 09h*4
                mov ax, word ptr cs:oldisr09
                mov [bx], ax
                mov ax, word ptr cs:oldisr09+2
                mov [bx+2], ax

                mov bx, 10h*4
                mov ax, word ptr cs:oldisr10
                mov [bx], ax
                mov ax, word ptr cs:oldisr10+2
                mov [bx+2], ax

                mov bx, 2fh*4
                mov ax, word ptr cs:oldisr2f
                mov [bx], ax
                mov ax, word ptr cs:oldisr2f+2
                mov [bx+2], ax

                mov byte ptr cs: freemem, 01            ;free memory
                jmp @@done

@@error:call cannot_remove_msg
                mov byte ptr cs: remove_flag, 00
@@done:         popa
                .pops
                sti
                ret
                endp

;-------------------------------------------------------------
cannot_remove_msg proc

                push cs
                pop ds
                mov dx, offset cant_rem_msg
                mov ah, 09h
                int 21h

                ret
cant_rem_msg    db 'Sorry, MSGShower cannot remove itself from RAM', 0dh, 0ah
                db 'Some programs hooked interrupt vectors after MSGShower', 0dh, 0ah
                db 'Remove this programs from RAM and try again'
                db '$'
                endp

;-------------------------------------------------------------
include         cmd.asm
;-------------------------------------------------------------
res_end:

mux_initproc

                call copyright                  ;search my copy in memory
                mov ah, mux_id
                int 2fh
                cmp bh, mux_id
                jne check_vmode

                call check_cmdline              ;check command line
```

```
                cmp cl, 0
                jne cmd

                call error_msg_01

cmd:            jmp terminate

;---------------------------------------------------------
check_vmode:    mov ah, 0fh                     ;check current video mode
                int 10h
                cmp al, 03
                je mux_ok
                call error_msg_02
                jmp terminate

mux_ok:         ret
                endp

;---------------------------------------------------------
resident_init   proc

                call enable_26line              ;initialization
                call get_smb
                call modify_cg
                call put_first_smb
;---------------------------------------------------------
                call ipxInit                    ;resident IPX init
                mov byte ptr cs: ipx_work, 00
                mov byte ptr cs: ipx_count, 00
;---------------------------------------------------------
                xor ax, ax                      ;save old vectors
                mov ds, ax
                mov ax, cs
                mov es, ax

                mov si, 08h*4
                mov di, offset oldisr08
                movsw
                movsw

                mov si, 09h*4
                mov di, offset oldisr09
                movsw
                movsw

                mov si, 10h*4
                mov di, offset oldisr10
                movsw
                movsw

                mov si, 2fh*4
                mov di, offset oldisr2f
                movsw
                movsw

                mov si, 28h*4
                mov di, offset oldisr28
                movsw
                movsw

;---------------------------------------------------------
                xor ax, ax                      ;hook vectors
                mov ds, ax

                mov bx, 08h*4
                cli
                mov word ptr [bx], offset new08
                mov ax, cs
                mov [bx+2], ax
                sti

                mov bx, 09h*4
                cli
                mov word ptr [bx], offset new09
                mov [bx+2], ax
                sti

                mov bx, 10h*4
                cli
```

```
                mov word ptr [bx], offset new10
                mov [bx+2], ax
                sti

                mov bx, 2fh*4
                cli
                mov word ptr [bx], offset new2f
                mov [bx+2], ax
                sti

                mov bx, 28h*4
                cli
                mov word ptr [bx], offset new28
                mov [bx+2], ax
                sti

;---------------------------------------------------------
                mov bx, cs:[2ch]        ;free environment
                mov es, bx
                mov ah, 49h
                int 21h

;               int 09h
;               mov cx, 8*80d
;test:          int 08h
;               loop test
;               call remove
;               int 08h

                call ok_msg

                mov dx, offset res_end          ;located memory

                mov ax, dx                      ;save memory in line
                push cs
                pop es
                mov bx, 16d
                mov di, offset memory
                call hex_ax

                mov ax, word ptr cs: my_number
                mov bx, 16d
                mov di, offset id_num
                call hex_ax

                shr dx, 4                       ;stay resident
                inc dx
                mov ax, 3100h
                int 21h

terminate:      mov ax, 4c01h                   ;xtr exit
                int 21h

                endp

;---------------------------------------------------------
copyright       proc

                push cs
                pop ds
                mov dx, offset @@msg
                mov ah, 09h
                int 21h

                ret
@@msg           db 'MSGShow (c) Sunny, 2002-2003, ver 0.2', 0dh, 0ah
                db '$'
                endp

;---------------------------------------------------------
ok_msg          proc

                push cs
                pop ds
                mov dx, offset @@msg
                mov ah, 09h
                int 21h

                ;mov dx, offset msg_?
```

```
                ;mov ah, 09h
                ;int 21h

                ret
@@msg           db 'Successfuly installed...', 0dh, 0ah, '$'
                endp

                ret
                endp

;-------------------------------------------------------------
error_msg_01    proc

                push cs
                pop ds
                mov dx, offset @@msg
                mov ah, 09h
                int 21h

                ret
@@msg           db 'MSGShow ERROR 01: program already loaded', 0dh, 0ah
                db '$'
                endp

                ret
                endp

;-------------------------------------------------------------
error_msg_02    proc

                push cs
                pop ds
                mov dx, offset @@msg
                mov ah, 09h
                int 21h

                ret
@@msg           db 'MSGShow ERROR 02: wrong video mode', 0dh, 0ah
                db '$'
                endp

                ret
                endp

;-------------------------------------------------------------
end start
```

Файл MsgShow\CMD.ASM

```
;-------------------------------------------------------------
;cmd.asm, MSGShow ver 0.2, (c) Sunny, 2002-2003
;-------------------------------------------------------------

hex_ax          proc

;translates a number (word) into letters
;entry: es:di -  buffer (4 bytes), bx - operating sys (16d or 2d)

                push es si di dx

                cmp bx, 2d              ;проверка системы счисления
                je bin
                add di, 3
                jmp next
bin:            add di, 16

next:           xor dx, dx
                div bx

                push ax

                cmp dx, 0ah
                jae letter

                add dx, '0'
                jmp continue
letter:         sub dx, 10d
                add dx, 'A'
```

```
continue:       mov al, dl
                std
                stosb

                pop ax
                xor dx, dx

                cmp ax, 0
                je exit
                jmp next

exit:           pop dx di si es

                ret
                endp

;-------------------------------------------------------------
check_cmdline   proc

;exit: cl = 0 if empty cmdline

                mov bx, 80h
                mov cl, [bx]
                cmp cl, 00
                je @@done
                mov ah, '/'
                call next_arg
                call do_arg
                call next_arg
                cmp cl, 00
                je @@done
                call do_arg
                call next_arg
                cmp cl, 00
                je @@done
                call do_arg


@@done:         mov bx, 80h
                mov cl, [bx]
                ret
                endp

;-------------------------------------------------------------
do_arg          proc

                ;push ds dx ax                  ;cmd debug
                ;push cs
                ;pop ds
                ;mov byte ptr arg, al
                ;mov dx, offset msg
                ;mov ah, 09h
                ;int 21h
                ;pop ax dx ds

                push ax
                mov ah,  mux_id
                int 2fh
                pop ax

                ret
msg             db 'You have typed /'
arg             db 0
                db ' ', 0dh, 0ah, '$'
                endp

;-------------------------------------------------------------
next_arg proc

;Scans for next argument in cmdline
;entry: ah - separator, cl - cmdline length
;exit:  al - first matched char, ds:si offset of next arg
;dest: ax, cx

                mov si, cmd_ofs                 ;Cmd line offset
                cld
char:           lodsb                   ;AL <- cmd line byte
                cmp al, ah
```

```
                je cont                         ;Separator found
                cmp al, ' '
                je cont                         ;Space found
                cmp al, '       '
                je cont                         ;Tab found
                loop char
cont:           jmp after_ch

blank:          dec cl
after_ch:       lodsb                           ;AL <- cmd line byte
                cmp al, ah
                je blank                        ;Skip defined chars
                cmp al, ' '
                je blank                ;Skip spaces
                cmp al, '       '
                je blank                ;Skip tabs
                dec si
                mov cmd_ofs, si

                ret
cmd_ofs         dw 0081h                ;Current cmd line offset
                endp

;-------------------------------------------------------
```

Файл MsgShow\26LINE.ASM

```
;-------------------------------------------------------------
;26line.asm, MSGShow ver 0.2, (c) Sunny, 2002-2003
;-------------------------------------------------------------

.SaveReg        macro reg                       ; *CRTCRegData++ = CRTC[reg]
                mov ah, reg
                call read_crtc
                stosb
                endm

save_crtc       proc

;Stores values of CRTC regs

                pushf
                cli
                mov dx, cs:CRTCPort
                mov di, offset CRTCRegs
                cld

                .SaveReg 15h                    ; Vertical Blanking Start reg
                .SaveReg 12h                    ; Vertical Displayed reg
                .SaveReg 10h                    ; Vertical Retrace Start reg
                .SaveReg 09h                    ; Maximum Scan Lines reg

                popf
                ret
                endp

;-------------------------------------------------
.RestoreReg     macro reg                       ; CRTC[reg] = *CRTCRegData++
                lodsb
                mov ah, reg
                call write_crtc
                endm

restore_crtc    proc
;Restores values of CRTC regs

                pushf
                cli
                mov dx, cs:CRTCPort
                mov si, offset CRTCRegs
                cld

                .RestoreReg 15h                 ; Vertical Blanking Start reg
                .RestoreReg 12h                 ; Vertical Displayed reg
                .RestoreReg 10h                 ; Vertical Retrace Start reg

                popf
                ret
```

```
                endp

;-------------------------------------------------
.AdjustReg      macro reg                       ; CRTC[reg] += BL
                lodsb
                mov ah, reg
                call read_crtc
                add al, bl
                call write_crtc
                endm

adjust_crtc     proc

;Set 26 screen lines displayed

                pushf
                cli
                mov dx, cs:CRTCPort
                mov si, offset CRTCRegs
                cld

                mov bl, cs:CRTCReg09            ; Maximum Scan Line reg
                and bl, 01fh                    ; Only bits 0-4 valid
                inc bl                          ; BL = char height

                .AdjustReg 15h                  ; Vertical Blanking Start reg
                .AdjustReg 12h                  ; Vertical Displayed reg

                shr bl, 1                       ; 1/2 of height to center screen
                .AdjustReg 10h                  ; Vertical Retrace Start reg

                popf
                ret
                endp

;-------------------------------------------------
read_crtc       proc

;Read CRTC register
;Entry: ah = register number
;Exit:  al = register value

                xchg ah, al
                out dx, al                      ; Set register number
                inc dx
                xchg ah, al
                in al, dx                       ; Read value
                dec dx
                ret
                endp

;-------------------------------------------------
write_crtc      proc

;Write CRTC register
;Entry: ah = register number
;       al = register value

                xchg ah, al
                out dx, al                      ; Set register number
                inc dx
                xchg ah, al
                out dx, al                      ; Write value
                dec dx
                ret
                endp

;-------------------------------------------------
get_crtc_port   proc

                push ds
                xor ax, ax
                mov ds, ax
                mov ax, ds:[0463h]              ; BIOS CRTC port address
                pop ds
                mov cs:CRTCPort, ax

                ret
                endp
```

```
;-------------------------------------------------
enable_26line   proc

                .pushs

                call get_crtc_port
                call save_crtc
                call adjust_crtc

                .pops
                ret
                endp

;-------------------------------------------------
disable_26line  proc

                .pushs

                call restore_crtc

                .pops
                ret
                endp
;-------------------------------------------------
```

Файл MsgShow\SHIFT.ASM

```
;-------------------------------------------------------------
;shift.asm, MSGShow ver 0.2, (c) Sunny, 2002-2003
;-------------------------------------------------------------

get_smb         proc

;Gets current symbol from text and loads it's definition
;destr: ax bx cx dx

                .pushs

                mov bx, offset text
                cmp end_flag, 1
                je @@first
                add bx, word ptr text_curr
@@first:mov cl, byte ptr [bx]
                mov byte ptr smb, cl
                inc byte ptr text_curr
                mov byte ptr bit_curr, 0

                xor ch, ch
                shl cx, 4

                mov ax, 1130h
                mov bh, 6h
                push cx
                int 10h                         ;get addr std font
                pop cx
                add bp, cx                      ;offset of curr smb

                cld
                push es
                pop ds
                mov si, bp
                push cs
                pop es
                mov di, offset definition
                mov cx, 16/2
                rep movsw

                .pops
                ret
                endp

;-------------------------------------------------------------
modify_cg       proc

;Modifys buffer sumbol charactre generator
;destr: ax bx cx dx
```

```
                .pushs

                mov ax, 1100h
                mov bx, 1000h
                mov cx, 1
                mov dx, buff_smb
                mov bp, offset def_with_shift
                int 10h

                .pops
                ret
                endp

;---------------------------------------------------------

clear_def       proc
;Clears definition with shift
;destr: ax cx

                .pushs

                xor ax, ax
                mov di, offset def_with_shift
                mov cx, 16d/2
                rep stosw
                call modify_cg

                .pops
                ret
                endp

;---------------------------------------------------------
clear_line      proc

;Clears line on the screen
;destr: ax cx

                .pushs

                mov di, offset line
                mov cx, dlength
                xor al, al
                rep stosb

                mov ax, vmem
                mov es, ax
                mov ah, color
                mov di, dlength * line_num * 2
                mov cx, dlength * 2 / 2
                rep stosw

                .pops
                ret
                endp

;---------------------------------------------------------
put_lineproc

;Puts line on it's place on the screen
;destr: ax cx

                .pushs

                mov ax, vmem
                mov es, ax
                mov di, dlength * line_num * 2
                mov si, offset line
                mov cx, dlength
@@copy:         movsb
                inc di
                loop @@copy

                .pops
                ret
                endp

;---------------------------------------------------------
check_end       proc
```

```
;Checks end of text
;destr: bx

                mov bl, byte ptr text_curr
                mov bh, byte ptr text_len
                cmp bl, bh
                je @@end_of_txt

                mov word ptr end_flag, 0
                jmp @@done

@@end_of_txt:   mov word ptr end_flag, 1
                mov word ptr text_curr, 0

                mov byte ptr cs: ipx_count, 00
                call set_esr
                call ask_new_text
                call s_fill_blocks

@@done:         ret
                endp

;-------------------------------------------------------------
set_shift_num   proc
;Sets new shift_num before new symbol dispalying
;destr: bx

                mov bl, byte ptr cs: shift_num_new
                mov byte ptr cs: shift_num, bl

                ret
                endp

;-------------------------------------------------------------
put_first_smb   proc

;Puts first smb in line(byffer symbol)
;destr: ax bx

                .pushs
                call clear_def

                mov bx, vmem
                mov ds, bx
                mov bx, dlength * line_num * 2
                mov al, buff_smb
                mov ah, color
                mov word ptr [bx], ax
                mov byte ptr cs:line, al

                .pops
                ret
                endp
;-----------------------
-----------------------------------
put_smb         proc
;Puts symbol in current place in line and puts buffer after it
;desrtr: ax bx

                .pushs
                call clear_def

                mov di, word ptr place
                shl di, 1
                add di, dlength * line_num * 2

                mov bx, vmem
                mov es, bx
                mov bx, offset text
                cmp end_flag, 1
                je @@last
                add bx, word ptr text_curr
                dec bx
                dec bx
                jmp @@show
@@last:         add bx, word ptr text_len
                dec bx
@@show:         mov al, byte ptr [bx]
                mov ah, color
```

```
                mov bx, offset line
                add bx, word ptr place
                cld

                stosw
                mov byte ptr [bx], al
                inc bx

                mov al, buff_smb
                stosw
                mov byte ptr [bx], al

                inc byte ptr place

                .pops
                ret
                endp

;-------------------------------------------------------------
shift           proc

;Makes shift in symbol definition
;destr: ax bx cx

                .pushs

                call put_line

                mov di, offset def_with_shift
                mov si, offset definition
                mov cx, 16/2
                cld
                rep movsw

                mov al, 11111111b
                mov cl, 8
                sub cl, byte ptr bit_curr
                shl al, cl
                mov cx, 16d
                mov bx, offset def_with_shift
@@loop:         and byte ptr [bx], al
                inc bx
                loop @@loop

                mov bl, byte ptr shift_num
                add byte ptr bit_curr, bl

                .pops
                ret
                endp

;-------------------------------------------------------------
do              proc

                cmp byte ptr bit_curr, 8
                jne @@ok
                cmp word ptr place, dlength - 1
                jne @@last_bit

                call clear_line
                call set_shift_num      ;NEW!
                call check_end
                call get_smb
                call put_first_smb
                mov byte ptr bit_curr, 0
                mov word ptr place, 0
                jmp @@ok

@@last_bit:     call check_end
                call set_shift_num      ;NEW!
                call get_smb
                call put_smb
                mov byte ptr bit_curr, 0
@@ok:           call shift
                call modify_cg

                ret
                endp
;-------------------------------------------------------------
```

Файл MsgShow\IPXINT.ASM

```
;-------------------------------------------------------------
;from sender    - 00 (1 byte), <addr (6 bytes)> - addres of sender
;               - 01 (1 byte), <length (1 byte)>, <text(length bytes)> - next text line
;               - 02 (1 byte), <number (2 bytes)> - number of THIS node
;from reciver   - 03 (1 byte), <number (2 bytes)> - asking for next text line
;               - 01 (1 byte), <addr (6 bytes)> - new node with its addr
;               - 02 (1 byte), <number (2 bytes)> - kill node
;-------------------------------------------------------------

                db '<START OF IPX DATA>'
                db '<S Packet Header>'    ;SEND
sChecksum       db 2 dup (0)
sLength         db 2 dup (0)
sTrnsprt_cntrl  db 1 dup (0)
sPacket_Type    db 1 dup (0)
sDest_Network   db 4 dup (0)
sDest_Node      db 6 dup (0)
sDest_Socket    db 2 dup (0)
sSrc_Network    db 4 dup (0)
sSrc_Node       db 6 dup (0)
sSrc_Socket     db 2 dup (0)

;-------------------------------------------------------------
                db '<S ECB>'
sLink           db 4 dup (0)
sESR_Adr        db 4 dup (0)
sIn_Use         db 0
sCompl_Code     db 0
sSocket_Number  db 2 dup (0)
sIPX_Wrkspc     db 4 dup (0)
sDriver_Wrkspc  db 12 dup (0)
sImmediate_Adr  db 6 dup (0)
sFragment_Count db 2 dup (0)
sFrag1_Adr      db 4 dup (0)                    ;off,seg
sFrag1_Size     db 2 dup (0)
sFrag2_Adr      db 4 dup (0)
sFrag2_Size     db 2 dup (0)

;-------------------------------------------------------------
                db '<S Buffer>'
sBuffer         db packet_len dup (0)
;-------------------------------------------------------------

                db '<R Packet Header>'    ;RECIEVE
rChecksum       db 2 dup (0)
rLength         db 2 dup (0)
rTrnsprt_cntrl  db 1 dup (0)
rPacket_Type    db 1 dup (0)
rDest_Network   db 4 dup (0)
rDest_Node      db 6 dup (0)
rDest_Socket    db 2 dup (0)
rSrc_Network    db 4 dup (0)
rSrc_Node       db 6 dup (0)
rSrc_Socket     db 2 dup (0)

;-------------------------------------------------------------
                db '<R ECB>'
rLink           db 4 dup (0)
rESR_Adr        db 4 dup (0)
rIn_Use         db 0
rCompl_Code     db 0
rSocket_Number  db 2 dup (0)
rIPX_Wrkspc     db 4 dup (0)
rDriver_Wrkspc  db 12 dup (0)
rImmediate_Adr  db 6 dup (0)
rFragment_Count db 2 dup (0)
rFrag1_Adr      db 4 dup (0)                    ;off,seg
rFrag1_Size     db 2 dup (0)
rFrag2_Adr      db 4 dup (0)
rFrag2_Size     db 2 dup (0)

;-------------------------------------------------------------
                db '<R Buffer>'
rBuffer         db packet_len dup (0)
                db '<END OF IPX DATA>'
```

```
;-------------------------------------------------------------
set_esr         proc

                mov word ptr sESR_Adr, offset s_ESR
                mov cx, cs
                mov word ptr sESR_Adr + 2, cx

                mov word ptr cs:rESR_Adr, offset r_ESR
                mov cx, cs
                mov word ptr cs:rESR_Adr + 2, cx

                ret
                endp

;-------------------------------------------------------------
s_ESR           proc far

                ;.putsmb '*'

                .pushs
                pusha
                call set_esr
                call r_fill_blocks

                popa
                .pops

                retf
                endp

;-------------------------------------------------------------
r_ESR           proc far

                cmp byte ptr cs: rBuffer, 01
                je @@new_text

                .pushs
                pusha
                ;call cancel_all        ;NOT DEBUGGED!
                ;call ask_new_text              ;
                ;call set_esr                   ;
                ;call s_fill_blocks             ;
                ;.putsmb '%'
                popa
                .pops
                jmp @@done

@@new_text:     ;.putsmb '$'

                .pushs
                pusha
                call get_new_text

                mov byte ptr cs:ipx_work, 00
                popa
                .pops

@@done:         retf
                endp

;-------------------------------------------------------------
s_fill_blocks   proc

                ;.putsmb 'S'

                .pushs
                pusha

                push cs
                pop ds
                push cs
                pop es

                ;.putsmb '1'
                mov byte ptr cs:ipx_work, 01            ;IPX FLAG SET

                mov word ptr sDest_Network, 0000
                mov word ptr sDest_Network + 2, 0000
```

```
                mov si, offset sender_adr
                mov di, offset sDest_Node
                mov cx, 6d
                cld
                rep movsb

                ;.putsmb '2'
                mov word ptr sDest_Socket, socket

                cmp byte ptr cs: localt_init, 01
                je @@send

                mov byte ptr cs: localt_init, 01
                push cs
                pop es
                mov si, offset sDest_Network
                mov di, offset sImmediate_Adr
                ;.putsmb '3'
                ;.dumpmem sender_adr, 6
                call ipxGetLocalTarget

                ;.putsmb '4'
                mov word ptr sSocket_Number, socket
                mov word ptr sFragment_Count, 2d
                mov word ptr sFrag1_Adr, offset sChecksum
                mov word ptr sFrag1_Adr + 2, cs
                mov word ptr sFrag1_Size, 30d
                mov word ptr sFrag2_Adr, offset sBuffer
                mov word ptr sFrag2_Adr + 2, cs
                mov word ptr sFrag2_Size, packet_len
                mov word ptr sPacket_Type, ipx_packet_type

                ;.putsmb '5'
@@send:         push cs
                pop es
                mov si, offset sLink
                call ipxSendPacket

                cmp sCompl_Code, 00
                je @@done

                ;.putsmb '!'

@@done:         popa
                .pops
                ;.putsmb 'E'
                ret
localt_init     db 0
                endp

;---------------------------------------------------------------
r_fill_blocks   proc

                .pushs
                pusha

                mov byte ptr cs:ipx_work, 01

                push cs
                pop ds
                push cs
                pop es

                mov word ptr rSocket_Number, socket
                mov word ptr rFragment_Count, 2d
                mov word ptr rFrag1_Adr, offset rChecksum
                mov word ptr rFrag1_Adr + 2, cs
                mov word ptr rFrag1_Size, 30d
                mov word ptr rFrag2_Adr, offset rBuffer
                mov word ptr rFrag2_Adr + 2, cs
                mov word ptr rFrag2_Size, packet_len
                mov word ptr rPacket_Type, ipx_packet_type

                push cs
                pop es
                mov si, offset rLink
                call ipxListenForPacket

                popa
```

```
                .pops
                ret
                endp

;-------------------------------------------------------------
clr_buffer      proc

;entry: al - number of buffer (0 - recieve, 1 - send)

                push ax cx di es

                cmp al, 0
                jne @@clr_sbuf
                mov di, offset sBuffer
                jmp @@clr

@@clr_sbuf:     mov di, offset rBuffer
@@clr:          push cs
                pop es
                xor al, al
                mov cx, packet_len
                cld
                rep stosb

                pop es di cx ax
                ret
                endp

;-------------------------------------------------------------
cancel_all      proc

                push es si
                push cs
                pop es

                mov si, offset sLink
                call ipxCancelEvent
                mov si, offset rLink
                call ipxCancelEvent
                pop si es

                ret
                endp

;-------------------------------------------------------------
get_new_text    proc

                push es ds cx si di ax

                push cs
                pop es
                push cs
                pop ds
                mov cl, byte ptr rBuffer + 1
                xor ch, ch
                mov byte ptr cs: text_len, cl
                mov si, offset rBuffer + 2
                mov di, offset text
                rep movsb

                xor al, al
                call clr_buffer
                inc al
                call clr_buffer

                ;.putsmb '#'

                pop ax di si cx ds es
                ret
                endp

;-------------------------------------------------------------
ask_new_text    proc

                push  bx
                mov byte ptr cs:sBuffer, 03
                mov bx, word ptr cs:my_number
                mov word ptr cs:sBuffer + 1, bx
                pop bx
```

```
                ret
                endp

;-------------------------------------------------------------
ask_kill_node   proc

                push  bx
                mov byte ptr cs:sBuffer, 02
                mov bx, word ptr cs:my_number
                mov word ptr cs:sBuffer + 1, bx
                pop bx

                ret
                endp

;-------------------------------------------------------------
disconnect      proc

                call ask_kill_node
                call s_fill_blocks

                ret
                endp

;-------------------------------------------------------------
```

Файл MsgShow\IPXLIB.ASM

```
;-------------------------------------------------------------
;26line.asm, MSGShow ver 0.2, (c) Sunny, 2002-2003
;-------------------------------------------------------------

.ipxAPI         macro
                call dword ptr cs: entry_point
                endm

;-------------------------------------------------------------
ipxInit         proc

;IPX initialization
;exit: al - 00 if success

                mov ax, 7a00h
                int 2fh

                mov word ptr cs:entry_point, di
                mov word ptr cs:entry_point + 2, es

                ret
                endp

;-------------------------------------------------------------
ipxSocketOpen   proc

;entry: dx - socket, al - lengevity (00 - shortlived, ff - longlived)
;return: al - result (00 - OK, ff - already opened)
;        dx - socket number

                .pushs
                push bx

                xor bx, bx
                .ipxAPI

                pop bx
                .pops

                ret
                endp

;-------------------------------------------------------------
ipxGetLocalTarget proc

;Gets immediate addr of machine
;entry: es:si - addr (network, node) 10 bytes
;       es:di - 6 bytes for immediate adr
```

```
                push bx
                .pushs

                mov bx, 02h
                .ipxAPI

                cmp al, 00
                je @@done
                .putsmb '!'

@@done:         .pops
                pop bx

                ret
                endp

;-------------------------------------------------------------
ipxGetInternetworkAddress proc

;entry: es:si - 10 byte buffer for net and node number

                push bx
                .pushs

                mov bx, 09h
                .ipxAPI

                .pops
                pop bx

                ret
                endp

;-------------------------------------------------------------
ipxCancelEvent  proc

;entry: es:si - ECB block

                push bx
                .pushs

                mov bx, 06h
                .ipxAPI

                .pops
                pop bx

                ret
                endp

;-------------------------------------------------------------
ipxListenForPacket proc

;entry: es:si - filled ECB block

                .pushs
                push bx

                mov bx, 4
                .ipxAPI

;@@check_inuse: cmp In_Use, 00
;               je @@done
;               call ipxRelinquishControl
;               jmp @@check_inuse

@@done:         pop bx
                .pops
                ret
                endp

;-------------------------------------------------------------
ipxCheckInuseS  proc

@@check_inuse:  cmp sIn_Use, 00
                je @@done
                call ipxRelinquishControl
                jmp @@check_inuse
```

```
@@done:         ret
                endp

;---------------------------------------------------------
ipxCheckInuseR  proc

@@check_inuse:  cmp rIn_Use,00
                je @@done
                call ipxRelinquishControl
                jmp @@check_inuse

@@done:         ret
                endp

;---------------------------------------------------------
ipxSendPacket   proc

;entry: es:si - filled ECB block

                push bx
                .pushs

                mov bx, 03h
                .ipxAPI

                mov byte ptr ipx_work, 01

;@@check_inuse: cmp In_Use, 00
;               je @@done
;               call ipxRelinquishControl
;               jmp @@check_inuse


@@done:         .pops
                pop bx

                ret
                endp

;---------------------------------------------------------
ipxRelinquishControl proc

                .pushs
                push bx

                mov bx, 0ah
                .ipxAPI

                pop bx
                .pops
                ret
                endp

;---------------------------------------------------------
                db '<START OF IPX DATA>'
                db '<Driver entry point>'
entry_point     dd 0

;---------------------------------------------------------
                db '<Messages>'
msg_no_ipx      db 'IPX not present', '$'
msg_no_socket   db 'Cannot open socket', '$'

;---------------------------------------------------------
                db '<END OF IPX DATA>'
```

Файл MsgShow\ESR.ASM

```
;---------------------------------------------------------
;esr.asm, MSGShow ver 0.2, (c) Sunny, 2002-2003
;---------------------------------------------------------

.model tiny, C

.code
public  esrS
public  esrR
public  flagS
```

```
public  flagR

esrS            proc far

                ;push ds dx ax
                ;push cs
                ;pop ds
                ;mov dx, offset msgS
                ;mov ah, 09h
                ;int 21h
                ;pop ax dx ds

                mov word ptr cs: flagS, 01;

                retf
                msgS    db 'Send ESRve been executed', 0dh, 0ah, '$'
                flagS   dw 0
                endp

esrR            proc far

                ;push ds dx ax
                ;push cs
                ;pop ds
                ;mov dx, offset msgR
                ;mov ah, 09h
                ;int 21h
                ;pop ax dx ds

                mov word ptr cs: flagR, 01;

                retf
                msgR    db 'Recieve ESRve been executed', 0dh, 0ah, '$'
                flagR   dw 0
                endp

end
```

Файл MsgShow\MAIN_C.CPP

```
//-------------------------------------------------------------
//main_c.cpp, MSGShow ver 0.2, (c) Sunnym 2002-2003
//-------------------------------------------------------------

//#define DEBUG
//#define PACKETDUMP
//#define PACKETDUMPLEN 15
#define TEXTLEN 80
#define EXITKEY 27 //esc

extern "C"
        {
        void mux_init ();
        void resident_init ();
        extern int my_number;
        extern char sender_adr [6];
        extern char text [TEXTLEN];
        extern char text_len;
        }

#include <STDIO.h>
#include "esr.cpp"
#include "checkrr.cpp"
#include "ipx_init.cpp"


int main ()
        {
        mux_init ();
        printf ("IPX initialization. ");
        ipx_init ();
                #ifdef DEBUG
                printf ("my_number = %i\n", my_number);
                printf ("text_len = %i\n", (int)text_len);
                printf ("text = %s\n", text);
                #endif
        printf ("Staying resident. ");
        resident_init ();
```

```
        return 1;
        }
```

Файл MsgShow\IPX.H

```
//=============================================================================
// IPX.H
//=============================================================================

#ifndef __IPX_H
#define __IPX_H


typedef unsigned char byte;

enum PacketType
        {
        UNKNOWN_PACKET_TYPE      = 0x00,
        ROUTING_INFO_PACKET_TYPE = 0x01,
        ECHO_PACKET_TYPE         = 0x02,
        ERROR_PACKET_TYPE        = 0x03,
        IPX_PACKET_TYPE          = 0x04,
        SPX_PACKET_TYPE          = 0x05
        };

enum SocketAssignMode
        {
        SOCKET_DYNAMIC_ASSIGN    = 0x00,
        SOCKET_SHORT_LIVED       = 0x00,
        SOCKET_LONG_LIVED        = 0xFF
        };

enum IPXError
        {
        IPX_SUCCESS              = 0x00,
        EVENT_CANCELED           = 0xFC,
        FRAGMENT_SIZE_ERROR      = 0xFD,

        SOCKET_TABLE_FULL        = 0xFE, // IPXSocketOpen()
        SOCKET_ALREADY_OPEN      = 0xFF,

        DEST_NODE_PATH_NOT_FND   = 0xFA, // IPXGetInternetworkAddress()

        ECB_SOCKET_NOT_OPEN      = 0xFF, // IPXListenForPacket()

        DESTINATION_NOT_FOUND    = 0xFE, // IPXSendPacket()
        NETWORK_FAILURE          = 0xFF,

        CANNOT_CANCEL_ECB        = 0xF9, // IPXCancelEvent()
        ECB_NOT_IN_USE           = 0xFF
        };

//=============================================================================

struct IMMEDIATEADDR
{
byte data[6];
};

struct INTNETADDR
{
struct { byte data[4]; } network;
struct { byte data[6]; } node;
};

struct LOCALTARGET
{
INTNETADDR    intAddr;
IMMEDIATEADDR immediateAddr;
};

struct NETADDR
{
INTNETADDR intAddr;
unsigned   socket;
};
```

```
struct IPXHEADER
{
unsigned chksumHiLo;
unsigned lenHiLo;
byte     transportCtrl;
byte     packetType;
NETADDR  destAddr;
NETADDR  srcAddr;
};

struct ECBFRAGMENT
{
void far *buffer;
unsigned length;
};

struct ECB
{
void far      *linkAddr;
void far      (*ESR)(void);
byte          inUse;
byte          completionCode;
unsigned      socket;
byte          ipxWorkspace[4];
byte          drvWorkspace[12];
IMMEDIATEADDR immediateAddr;
unsigned      fragmentCount;
ECBFRAGMENT   fragment[2];
};

/*************************************************************************/

unsigned ipxInit();
unsigned ipxSocketOpen (byte *socket, byte longevity);
void     ipxSocketClose (byte *socket);
void     ipxSendPacket (ECB far *ecb);
void     ipxListenForPacket (ECB far *ecb);
void     ipxGetInternetworkAddress (INTNETADDR far *intaddr);
void     ipxRelinquishControl();
void     ipxDisconnectFromTarget (NETADDR far *addr);
unsigned ipxCancelEvent (ECB far *ecb);
void     ipxScheduleEvent (ECB far *ecb, unsigned delay);
unsigned ipxGetIntervalMarker();
unsigned ipxGetLocalTarget (LOCALTARGET far *lt);

#define HILOSWAP(x) ((x << 8) | (char)(x >> 8))

const int MINIPXDATALEN = 512,
          MAXIPXDATALEN = 4096;

void ipxShowECB (ECB *ecb, const char *prompt = NULL);
void ipxShowIPXPacket (IPXHEADER *ipxhdr);

const char *ipxReturnMsg (int val);
const char *ipxErrnoMsg (int val);


#endif
```

Файл MsgShow\IPX.CPP

```
//=========================================================================
// IPX.C
//=========================================================================

#include <mem.h>
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "ipx.h"

//=========================================================================

//static void far ErrorPrint();
```

```
static void (far* IPXAPI)() = NULL; //ErrorPrint;

#ifdef IPX_CPRINTF
#define PRINTF  cprintf
#else
#define PRINTF  printf
#endif


//=========================================================================

unsigned ipxInit()
{
_AX = 0x7a00;
geninterrupt (0x2f);
_SI = _AX;

((int*)&IPXAPI)[0] = _DI;
((int*)&IPXAPI)[1] = _ES;

_AX = _SI;
return _AL == 0xff;
}

//=========================================================================

unsigned ipxSocketOpen (unsigned *socket, int longevity)
{
_DX = *socket;
_AL = longevity;
_BX = 0x0000;
IPXAPI();
*socket = _DX;
_AH = 0;
return _AX;
}

//=========================================================================

void ipxSocketClose (unsigned socket)
{
_DX = socket;
_BX = 0x0001;
IPXAPI();
}

//=========================================================================

unsigned ipxGetLocalTarget (LOCALTARGET far *lt)
{
_ES = FP_SEG (lt);
_SI = FP_OFF (&lt->intAddr.network);
_DI = FP_OFF (&lt->immediateAddr);
_BX = 0x0002;
IPXAPI();
return _AL;
}

//=========================================================================

void ipxSendPacket (ECB far *ecb)
{
_ES = FP_SEG (ecb);
_SI = FP_OFF (ecb);
_BX = 0x0003;
IPXAPI();
}

//=========================================================================

void ipxListenForPacket (ECB far *ecb)
{
_ES = FP_SEG (ecb);
_SI = FP_OFF (ecb);
_BX = 0x0004;
IPXAPI();
}

//=========================================================================
```

```
void ipxGetInternetworkAddress (INTNETADDR far *intaddr)
{
_ES = FP_SEG (intaddr);
_SI = FP_OFF (intaddr);
_BX = 0x0009;
IPXAPI();
}

//============================================================================

void ipxRelinquishControl()
{
_BX = 0x000a;
IPXAPI();
}

//============================================================================

void ipxDisconnectFromTarget (NETADDR far *naddr)
{
_ES = FP_SEG (naddr);
_SI = FP_OFF (naddr);
_BX = 0x000b;
IPXAPI();
}

//============================================================================

unsigned ipxCancelEvent (ECB far *ecb)
{
_ES = FP_SEG (ecb);
_SI = FP_OFF (ecb);
_BX = 0x0006;
IPXAPI();
return _AL;
}

//============================================================================

unsigned ipxGetIntervalMarker()
{
_BX = 0x0008;
IPXAPI();
return _AX;
}

//============================================================================

void ipxScheduleEvent (ECB far *ecb, unsigned delay)
{
_ES = FP_SEG (ecb);
_SI = FP_OFF (ecb);
_AX = delay;
_BX = 0x0005;
IPXAPI();
return;
}

//============================================================================

void printb (const char *prompt, void *buffer, unsigned size)
{
const char *nl = strpbrk (prompt, "\n\r");
if (nl) PRINTF ("%.*s", nl-prompt, prompt);
else    PRINTF ("%s", prompt);

char *buf = (char*) buffer;
for (int i = 0; i < size; i++) PRINTF ("%02X ", (unsigned char)buf[i]);

if (nl) PRINTF ("\r\n");
}

void ipxShowECB (ECB *ecb, const char *prompt)
{
_setcursortype (_NOCURSOR);

if (prompt) PRINTF ("%s:\r\n", prompt);
```

```
PRINTF ("\r\n");
PRINTF ("Link addr: %Fp\r\n",  ecb->linkAddr);
PRINTF ("      ESR: %Fp\r\n",  ecb->ESR);
PRINTF ("   In use: %02X\r\n", ecb->inUse);
PRINTF ("Comp code: %02X\r\n", ecb->completionCode);
PRINTF ("   Socket: %04X\r\n", ecb->socket);
PRINTF ("\r\n");
printb ("Workspace: ",     ecb->ipxWorkspace,   sizeof (ecb->ipxWorkspace));
printb ("-- \n\r",         ecb->drvWorkspace,   sizeof (ecb->drvWorkspace));
printb ("Immd addr: \r\n", &ecb->immediateAddr, sizeof (ecb->immediateAddr));
PRINTF ("\r\n");
PRINTF (" Frag cnt: %u\r\n", ecb->fragmentCount);
for (int i = 0; i < ecb->fragmentCount; i++)
   PRINTF ("   Frag %03u - buf %Fp, len: %u\r\n",
                  i+1, ecb->fragment[i].buffer, ecb->fragment[i].length);
PRINTF ("\r\n");

_setcursortype (_NORMALCURSOR);
}

/*****************************************************************************/

void ipxShowIPXPacket (IPXHEADER *header, const char *prompt)
{
_setcursortype (_NOCURSOR);

if (prompt) PRINTF ("%s:\r\n", prompt);

PRINTF ("\r\n");
PRINTF ("    Checksum: %04X\r\n", HILOSWAP (header->chksumHiLo));
PRINTF (" IPX Pkt Len: %u\r\n",   HILOSWAP (header->lenHiLo));
PRINTF (" RX Data Len: %u\r\n",   HILOSWAP (header->lenHiLo) - sizeof (IPXHEADER));
PRINTF ("TransprtCtrl: %02X\r\n", header->transportCtrl);
PRINTF ("  PacketType: %02X\r\n", header->packetType);
PRINTF ("\r\n");

printb ("  SrcNetwork: ",         &header->srcAddr.intAddr.network,
                           sizeof (header->srcAddr.intAddr.network));

printb ("  SrcNode: ",            &header->srcAddr.intAddr.node,
                           sizeof (header->srcAddr.intAddr.node));

PRINTF ("  SrcSocket: %04X\r\n",  header->srcAddr.socket);

printb (" DestNetwork: ",         &header->destAddr.intAddr.network,
                           sizeof (header->destAddr.intAddr.network));

printb (" DestNode: ",            &header->destAddr.intAddr.node,
                           sizeof (header->destAddr.intAddr.node));

PRINTF (" DestSocket: %04X\r\n",  header->destAddr.socket);

_setcursortype (_NORMALCURSOR);
}

//===========================================================================

const char *ipxErrorMsg (int val)
{
const struct { int val; const char *msg; } errnoMsg[] =
   {{DEST_NODE_PATH_NOT_FND, "DEST_NODE_PATH_NOT_FND"},
        {EVENT_CANCELED,          "EVENT_CANCELED"},
        {FRAGMENT_SIZE_ERROR,     "FRAGMENT_SIZE_ERROR"},
        {DESTINATION_NOT_FOUND,   "DESTINATION_NOT_FOUND"},
        {SOCKET_TABLE_FULL,       "SOCKET_TABLE_FULL"},
        {ECB_SOCKET_NOT_OPEN,     "ECB_SOCKET_NOT_OPEN"},
        {NETWORK_FAILURE,         "NETWORK_FAILURE"},
        {SOCKET_ALREADY_OPEN,     "SOCKET_ALREADY_OPEN"}};

for (int i = 0; i < sizeof (errnoMsg) / sizeof (*errnoMsg); i++)
        if (errnoMsg[i].val == val) return errnoMsg[i].msg;
return NULL;
}

//===========================================================================

/*
static void far ErrorPrint()
{
```

```
fprintf (stderr, "IPX Library: ipxInit() not called\r\n");
abort();
}
*/
```

Файл MsgShow\IPX_INIT.CPP

```
//-------------------------------------------------------------
//ipx_init.cpp, MSGShow ver 0.2, (c) Sunnym 2002-2003
//-------------------------------------------------------------

int checkKbd ()
        {
        if (kbhit ()) if (getch () == EXITKEY) exit (1);
        }


void checkR ()
        {
        #ifdef DEBUG
        printf ("IPX PACKET\n");
        printf ("PACKET CODE: %i\n", (int)bufferR[0]);
        #endif
    iterator++;
        check_code_show ();
        if (iterator == 50)
                {
                send ();
                exit (1);
                };
        }

void ipx_init ()
        {

        initIPX ();
        unsigned time = ipxGetIntervalMarker ();
        recieve ();
        while (1)//(!checkKbd ())

                {
                //checkKbd ();
                if (flagR)
                        {
                        #ifdef DEBUG
                        printf ("flagR is nonzero\n");
                        #endif
                        flagR = 0;
                        if (!bufferR[0])
                                {
                                #ifdef DEBUG
                                printf ("SAP recieved\n");
                                #endif
                                checkR ();
                                send ();
                                time = ipxGetIntervalMarker ();
                                break;
                                }
                        };
                }
        send ();
        time = ipxGetIntervalMarker ();

        char lastcode = 0x00;
        while (!text_len)
                {
                //checkKbd ();
                if (checkTimeout (time))
                        {
                        #ifdef DEBUG
                        printf ("Timeout\n");
                        #endif
                        cancelAll ();
                        flagR = flagS = 0;
                        send ();
                        time = ipxGetIntervalMarker ();
                        }
                else
```

```
                        {
                        if (flagS)
                                {
                                #ifdef DEBUG
                                printf ("flagS is nonzero\n");
                                #endif
                                flagS = 0;
                                recieve ();
                                };
                        if (flagR)
                                {
                                #ifdef DEBUG
                                printf ("flagR is nonzero\n");
                                #endif
                                if ((lastcode == 0x00 && bufferR[0] == 0x02) ||
                                        (lastcode == 0x02 && bufferR[0] == 0x01) )
                                        {
                                        lastcode = bufferR[0];
                                        flagR = 0;
                                        checkR ();
                                        send ();
                                        time = ipxGetIntervalMarker ();
                                        }
                                else
                                        {
                                        cancelAll ();
                                        flagR = flagS = 0;
                                        send ();
                                        time = ipxGetIntervalMarker ();
                                        };
                                };
                        };
                };
        };
```

Файл MsgShow\ESR.CPP

```
//-------------------------------------------------------------
//esr.cpp, MSGShow ver 0.2, (c) Sunnym 2002-2003
//-------------------------------------------------------------

//-EXTERN-----------------------------------------------------------------------

extern "C"
        {
        void far esrS ();
        void far esrR ();
        extern int flagS;
        extern int flagR;
        }

//-INCLUDES---------------------------------------------------------------------

#include "ipx.cpp"

//-CONSTANTS--------------------------------------------------------------------

const int ESC                   = 27;
const int BUFFERLEN             = 83;
const int NODEADDRLEN           = 6;
const int NETADDRLEN            = 4;
const unsigned IPXTIMEOUT       = 30;
const int SOCKETNUM             = 0x0420;

//-PROTOTYPES-------------------------------------------------------------------

void recieve ();
void checkR ();

//-GLOBAL-DATA------------------------------------------------------------------


char iterator = 0;

ECB ecbR = {0};
IPXHEADER ipxHeaderR = {0};
char bufferR [BUFFERLEN] = {0};
```

```
ECB ecbS = {0};
IPXHEADER ipxHeaderS = {0};
char bufferS [BUFFERLEN] = {0};

char destadr [6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};

//-IPX-WORK-PROC'S-----------------------------------------------------------

int checkTimeout (unsigned lasttime)
        {
        return ipxGetIntervalMarker () - lasttime >= IPXTIMEOUT;
        }

void cancelAll ()
        {
        if (ipxCancelEvent(&ecbR) == 0xF9) printf ("ipxCancelEvent(&esrR) - unable to close\n");
        if (ipxCancelEvent(&ecbS) == 0xF9) printf ("ipxCancelEvent(&esrS) - unable to close\n");
        }

void clrAll ()
        {
        for (int i = 0; i < BUFFERLEN; i++)
                { bufferS[i] = 0x00; bufferR[i] = 0x00; }
        }

//-SEND-RECIEVE-PROC'S-------------------------------------------------------

int initIPX ()
        {
        if (!ipxInit())
                {
                printf ("IPX not present\n");
                return 0;
                };

        unsigned socket = SOCKETNUM;
        ipxSocketOpen (&socket, SOCKET_SHORT_LIVED);

        return 1;
        };

void send ()
        {
        for (int i = 0; i <  NETADDRLEN; i++) ipxHeaderS.destAddr.intAddr.network.data[i] = 0x00;
        for (    i = 0; i < NODEADDRLEN; i++) ipxHeaderS.destAddr.intAddr.node.data[i] =
sender_adr[i];
        ipxHeaderS.destAddr.socket = SOCKETNUM;

        LOCALTARGET localTarget = {0};
        localTarget.intAddr = ipxHeaderS.destAddr.intAddr;
        if ((errno = ipxGetLocalTarget (&localTarget)) != IPX_SUCCESS)
                printf ("ipxGetLocalTarget() error 0x%02X (%s)\n", errno, ipxErrorMsg (errno));

        ecbS.socket             = SOCKETNUM;
        ecbS.immediateAddr      = localTarget.immediateAddr;
        ecbS.fragmentCount      = 2;
        ecbS.fragment[0].buffer = &ipxHeaderS;
        ecbS.fragment[0].length = sizeof (ipxHeaderS);
        ecbS.fragment[1].buffer = bufferS;
        ecbS.fragment[1].length = sizeof (bufferS);
        ecbS.ESR                = (void(far*)()) MK_FP (_CS, esrS);

        ipxSendPacket (&ecbS);
        ipxRelinquishControl ();
        #ifdef DEBUG
        printf ("Packet've been sent\n");
        #endif
        }

void recieve ()
        {
        ecbR.socket             = SOCKETNUM;
        ecbR.fragmentCount      = 2;
        ecbR.fragment[0].buffer = &ipxHeaderR;
        ecbR.fragment[0].length = sizeof (ipxHeaderR);
        ecbR.fragment[1].buffer = bufferR;
        ecbR.fragment[1].length = sizeof (bufferR);
        ecbR.ESR                = (void(far*)()) MK_FP (_CS, esrR);
```

```
        ipxListenForPacket (&ecbR);
        ipxRelinquishControl ();

        #ifdef DEBUG
        printf ("Listening for packet...\n");
        #endif
        }
```

Файл MsgShow\CHECKRR.CPP

```
//-------------------------------------------------------------
//checkrr.cpp, MSGShow ver 0.2, (c) Sunnym 2002-2003
//-------------------------------------------------------------

void fromsend_00 ()
        {
        #ifdef PACKETDUMP
        printf ("00 from send (send addr)\n");
        printf ("BUFFERR: ");
        for (int j = 0; j < PACKETDUMPLEN; j++) printf ("x%02x ", bufferR[j]);
        printf ("\n");
        #endif PAKETDUMP

        for (int i = 0; i < NODEADDRLEN; i++)
                sender_adr[i] = bufferR[1 + i];
        char buf [10] = {0};
        ipxGetInternetworkAddress ((INTNETADDR*) buf);
        clrAll ();
        bufferS[0] = 0x01;
        for (i = 0; i < NODEADDRLEN; i++)
                bufferS[1 + i] = buf [4 + i];

        #ifdef PACKETDUMP
        printf ("BUFFERS: ");
        for (j = 0; j < PACKETDUMPLEN; j++) printf ("x%02x ", bufferS[j]);
        printf ("\n");
        #endif PAKETDUMP
        }

void fromsend_01 ()
        {
        #ifdef PACKETDUMP
        printf ("01 from send (new text)\n");
        printf ("BUFFERR: ");
        for (int j = 0; j < PACKETDUMPLEN; j++) printf ("x%02x ", bufferR[j]);
        printf ("\n");
        #endif PAKETDUMP

        text_len = *(int*)(&bufferR[1]);
        for (int i = 0; i < text_len; i++)
                text[i] = bufferR[2 + i];
        clrAll ();

        #ifdef PACKETDUMP
        printf ("BUFFERS: ");
        for (j = 0; j < PACKETDUMPLEN; j++) printf ("x%02x ", bufferS[j]);
        printf ("\n");
        #endif PAKETDUMP
        }

void fromsend_02 ()
        {
        #ifdef PACKETDUMP
        printf ("02 from send (my ID number)\n");
        printf ("BUFFERR: ");
        for (int j = 0; j < PACKETDUMPLEN; j++) printf ("x%02x ", bufferR[j]);
        printf ("\n");
        #endif PAKETDUMP

        my_number = *(int*)(&bufferR[1]);
        clrAll ();
        bufferS[0] = 0x03;
        bufferS[1] = *(char*)(&my_number);

        #ifdef PACKETDUMP
        printf ("BUFFERS: ");
        for (j = 0; j < PACKETDUMPLEN; j++) printf ("x%02x ", bufferS[j]);
        printf ("\n");
```

```
        #endif PAKETDUMP
        }

void check_code_show ()
        {
        switch (bufferR[0])
                {
                case 0x00: fromsend_00 (); break;
                case 0x01: fromsend_01 (); break;
                case 0x02: fromsend_02 (); break;
                };
        }
```

Файл MsgSend\CADR.CPP

```
//-----------------------------------------------------------
//cadr.cpp, MSGSend ver 0.03, (c) Sunny, 2002-2003
//-----------------------------------------------------------

typedef unsigned char byte;

#define DEFADRLEN 6
class CAdr : public CObj
        {
        private:
        int myLength;

        public:
        byte* myAdr;

        CAdr (byte* adr, int len = DEFADRLEN)
                {
                myLength = len;
                myAdr = new byte [myLength];
                for (int i = 0; i < myLength; i++) myAdr[i] = *(adr+i);
                };
        CAdr ()
                {
                myLength = DEFADRLEN;
                myAdr = new byte [myLength];
                for (int i = 0; i < myLength; i++) myAdr[i] = '?';
                };
        CAdr (int len)
                {
                myLength = len;
                myAdr = new byte [myLength];
                for (int i = 0; i < myLength; i++) myAdr[i] = '?';
                };
        ~CAdr () { delete [] myAdr; };

        void operator = (byte* adr)
                { for (int i = 0; i < myLength; i++) myAdr[i] = *(adr+i);       };

        void operator = (char* adr)
                { for (int i = 0; i < myLength; i++) myAdr[i] = *(adr+i);       };

        void operator = (CAdr& adr)
                { for (int i = 0; i < myLength; i++) myAdr[i] = adr.myAdr[i]; };

        int operator == (CAdr& adr)
                {
                for (int i = 0; i < myLength; i++)
                        if (myAdr[i] != adr.myAdr[i]) return 0;
                return 1;
                };

        int operator != (CAdr& adr)
                {
                for (int i = 0; i < myLength; i++)
                        if (myAdr[i] != adr.myAdr[i]) return 1;
                return 0;
                };

        virtual void dump (int tab)
                {
                for (int i = 0; i < myLength; i++) fprintf (myOutput, "\x%02x ", myAdr[i]);
                fprintf (myOutput, "\n");
                };
```

```
        };
```

Файл MsgSend\CINFO.CPP

```
//-----------------------------------------------------------
//cinfo.cpp, MSGSend ver 0.03, (c) Sunny, 2002-2003
//-----------------------------------------------------------

//-CRec-------------------------------------------------------------------------

#define MAXRECLEN 80
class CRec : public CObj
        {
        public:

        int myLen;
        char myText [MAXRECLEN+1];

        CRec  () { myLen = 0; myText[0] = '\0'; };
        ~CRec () { };

        int addSymbol (char ch)
                {
                if (myLen == MAXRECLEN+1) return 0;
                myText[myLen] = ch;
                myLen++;
                myText[myLen] = '\0';
                return 1;
                };

        virtual void dump (int tab = 0)
                {
                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "CRec:\n");
                putTab (tab);   fprintf (myOutput, "{\n");
                putTab (tab+1); fprintf (myOutput, "myLen = %i\n", myLen);
                putTab (tab+1); fprintf (myOutput, "myText = %s", myText);
                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "};");
                };
        };
//-CInfo------------------------------------------------------------------------

class CInfo : public CObj
        {
        private:
        CList<CRec> myRecs;

        void emptyRec (CRec& rec);

        public:
        CInfo () {};
        ~CInfo () {};

        int readFile (const char* name);
        int getNumberOfRecs ();
        CRec& operator [] (int num);

        virtual void dump (int tab = 0);
        };

//-----------------------------------------------------------------------------

void CInfo::emptyRec (CRec& rec)
        {
        rec.myLen = 0;
        rec.myText[0] = '\0';
        };

//-----------------------------------------------------------------------------

int CInfo::readFile (const char* name)
        {
        FILE* file;
        if (!(file = fopen (name, "r"))) return 0;
        CRec rec;

        #define ever (;;)
```

```
        for ever
                {
                rec.addSymbol (getc (file));
                if (rec.myText[rec.myLen - 1] == '\n')
                        {
                        rec.myLen--; rec.myText[rec.myLen] = '\0';
                        myRecs.addUnit (rec); emptyRec (rec);
                        };
                if (rec.myLen == MAXRECLEN)
                        {
                        myRecs.addUnit (rec);
                        emptyRec (rec);
                        };
                if (feof(file))
                        {
                        if (rec.myLen != 1) myRecs.addUnit (rec);
                        break;
                        };
                };
        fclose (file);
        return getNumberOfRecs ();
        };

//---------------------------------------------------------------------------

int CInfo::getNumberOfRecs ()
        { return myRecs.getNumberOfUnits() - 1; };

//---------------------------------------------------------------------------

CRec& CInfo::operator [] (int num)
        { return myRecs[num]; };

//---------------------------------------------------------------------------

void CInfo::dump (int tab)
        {
                        fprintf (myOutput, "\n");
        putTab (tab);   fprintf (myOutput, "CInfo:\n");
        putTab (tab);   fprintf (myOutput, "{\n");
        putTab (tab+1); fprintf (myOutput, "myRecs:"); myRecs.dump (tab+2);
                        fprintf (myOutput, "\n");
        putTab (tab);   fprintf (myOutput, "};");
        };
//---------------------------------------------------------------------------
```

Файл MsgSend\CLIST.CPP

```
//-----------------------------------------------------------
//clist.cpp, MSGSend ver 0.03, (c) Sunny, 2002-2003
//-----------------------------------------------------------

#include <STDIO.H>

//-ListUnit----------------------------------------------------------------

template <class Type>
struct Unit : public CObj
        {
        Type myThing;
        Unit* myNext;
        unsigned myNumber;
        Unit* myPrev;

        virtual void dump (int tab = 0)
                {
                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "Unit:\n");
                putTab (tab);   fprintf (myOutput, "{\n");
                putTab (tab+1); fprintf (myOutput, "myNumber = %u\n", myNumber);
                putTab (tab+1); fprintf (myOutput, "myThing: "); ::dump (myThing, tab + 2);
                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "};");
                }
        }

//-CList-------------------------------------------------------------------
```

```
template <class Type>
class CList : public CObj
        {
        private:
        int myNumberOfUnits;
        unsigned myNextNumber;
        Unit<Type> myFirstUnit;

        Unit<Type>* getLastPointer ();
        Unit<Type>* getPointer (unsigned num);

        public:
        CList (Type thing);
        CList ();
        ~CList ();

        int getNumberOfUnits ();
        unsigned addUnit (Type& thing);
        void delUnit (unsigned num);
        Type& operator [] (unsigned num);

        virtual void dump (int tab);
        };

//---------------------------------------------------------------------------

template <class Type>
Unit<Type>* CList<Type>::getLastPointer ()
        {
        Unit<Type>* next = &myFirstUnit;
        Unit<Type>* last = &myFirstUnit;
        while (next)
                {
                last = next;
                next = next->myNext;
                };
        return last;
        };

//---------------------------------------------------------------------------

template <class Type>
Unit<Type>* CList<Type>::getPointer (unsigned num)
        {
        Unit<Type>* next = &myFirstUnit;
        while (next->myNumber != num)
                if (next) next = next->myNext; else return NULL;
        return next;
        };

//---------------------------------------------------------------------------

template <class Type>
CList<Type>::CList (Type thing)
        {
        myFirstUnit.myThing = thing;
        myFirstUnit.myNext = NULL;
        myFirstUnit.myPrev = NULL;
        myFirstUnit.myNumber = 0;
        myNextNumber = 1;
        myNumberOfUnits = 1;
        };

//---------------------------------------------------------------------------

template <class Type>
CList<Type>::CList ()
        {
        myFirstUnit.myNext = NULL;
        myFirstUnit.myPrev = NULL;
        myFirstUnit.myNumber = 0;
        myNextNumber = 1;
        myNumberOfUnits = 1;
        };

//---------------------------------------------------------------------------

template <class Type>
CList<Type>::~CList ()
```

```
        {
        Unit<Type>* next = myFirstUnit.myNext;
        while (next)
                {
                Unit<Type>* kill = next;
                next = next->myNext;
                delete kill;
                };
        };

//---------------------------------------------------------------------------

template <class Type>
int CList<Type>::getNumberOfUnits ()
        { return myNumberOfUnits; };

//---------------------------------------------------------------------------

template <class Type>
unsigned CList<Type>::addUnit (Type& thing)
        {
        Unit<Type>* last = getLastPointer ();
        Unit<Type>* prev = last;
        last->myNext = new Unit<Type>;
        if (!last->myNext) return 0;
        last = last->myNext;
        last->myPrev = prev;
        last->myNext = NULL;
        last->myThing = thing;
        last->myNumber = myNextNumber;
        myNextNumber++;
        myNumberOfUnits++;
        return last->myNumber;
        };

//---------------------------------------------------------------------------

template <class Type>
void CList<Type>::delUnit (unsigned num)
        {
        Unit<Type>* thisPointer = getPointer (num);
        Unit<Type>* prev = thisPointer->myPrev;
        Unit<Type>* next = thisPointer->myNext;
        prev->myNext = thisPointer->myNext;
        next->myPrev = thisPointer->myPrev;
        myNumberOfUnits--;
        delete thisPointer;
        }

//---------------------------------------------------------------------------

template <class Type>
Type& CList<Type>::operator[] (unsigned num)
        {
        Unit<Type>* next = &myFirstUnit;
        while (next->myNumber != num)
                {
                next = next->myNext;
                if (!next) return myFirstUnit.myThing;
                };
        return next->myThing;
        };

//---------------------------------------------------------------------------

template <class Type>
void CList<Type>::dump (int tab = 0)
                {
                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "CList:\n");
                putTab (tab);   fprintf (myOutput, "{\n");
                putTab (tab+1); fprintf (myOutput, "myNumberOfUnits = %i\n", myNumberOfUnits);
                putTab (tab+1); fprintf (myOutput, "myNextNumber = %u\n", myNextNumber);
                putTab (tab+1); fprintf (myOutput, "myList: ");

                Unit<Type>* next = &myFirstUnit;
                while (next)
                        {
                        ::dump (next, tab + 1);
```

```
                        next = next->myNext;
                        };

                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "};");
                }

//---------------------------------------------------------------------------
```

Файл MsgSend\CNET.CPP

```
//-----------------------------------------------------------
//cnet.cpp, MSGSend ver 0.03, (c) Sunny, 2002-2003
//-----------------------------------------------------------

#include "cadr.cpp"

//-CNode-------------------------------------------------------------------

class CNode : public CObj
        {
        public:
        CAdr myAdr;
        unsigned myCurRecord;

        CNode  () { myCurRecord = 1; };
        ~CNode () { };

        int operator == (CNode& node)
                {
                if (myAdr == node.myAdr && myCurRecord == node.myCurRecord) return 1;
                return 0;
                };

        int operator != (CNode& node)
                {
                if (myAdr != node.myAdr || myCurRecord != node.myCurRecord) return 1;
                return 0;
                };

        virtual void dump (int tab = 0)
                {
                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "CNode:\n");
                putTab (tab);   fprintf (myOutput, "{\n");
                putTab (tab+1); fprintf (myOutput, "myAdr = "); myAdr.dump (0);
                putTab (tab+1); fprintf (myOutput, "myCurRecord = %u", myCurRecord);
                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "};");
                };
        };

//-CNet--------------------------------------------------------------------

#define NETADRLEN 4

class CNet : public CObj
        {
        private:
        CAdr myAdr;
        CList<CNode> myNet;

        public:
        CNet  () : myAdr (NETADRLEN) {};
        ~CNet () {};

        void setNetAdr (byte* adr);
        byte* getNetAdr ();
        unsigned addNode (byte* adr);
        void delNode (unsigned num);
        int getCurRecord (unsigned num);
        void setCurRecord (unsigned num, int rec);
        byte* getNodeAdr (unsigned num);

        virtual void dump (int tab = 0);
        };

//---------------------------------------------------------------------------
```

```
void CNet::setNetAdr (byte* adr) { myAdr = adr; };

//---------------------------------------------------------------------------

byte* CNet::getNetAdr () { return myAdr.myAdr; };

//---------------------------------------------------------------------------

unsigned CNet::addNode (byte* adr)
        {
        CNode node;
        node.myAdr = adr;
        return myNet.addUnit (node);
        };

//---------------------------------------------------------------------------

void CNet::delNode (unsigned num)
        { myNet.delUnit (num); };

//---------------------------------------------------------------------------

int CNet::getCurRecord (unsigned num)
        { return myNet[num].myCurRecord; };

//---------------------------------------------------------------------------

void CNet::setCurRecord (unsigned num, int rec)
        { myNet[num].myCurRecord = rec; };

//---------------------------------------------------------------------------

byte* CNet::getNodeAdr (unsigned num)
        { return myNet[num].myAdr.myAdr; };

//---------------------------------------------------------------------------

void CNet::dump (int tab)
        {
                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "CNet:\n");
                putTab (tab);   fprintf (myOutput, "{\n");
                putTab (tab+1); fprintf (myOutput, "myAdr = "); myAdr.dump (0);
                putTab (tab+1); fprintf (myOutput, "myNet:"); myNet.dump (tab+2);
                                fprintf (myOutput, "\n");
                putTab (tab);   fprintf (myOutput, "};");
        };

//---------------------------------------------------------------------------
```

Файл MsgSend\COBJ.CPP

```
//-----------------------------------------------------------
//cobj.cpp, MSGSend ver 0.03, (c) Sunny, 2002-2003
//-----------------------------------------------------------

#include <STDIO.H>

//-CObj---------------------------------------------------------------------

class CObj
        {
        public:
        virtual void dump (int tab) {};
        };

//-ForDump------------------------------------------------------------------

#define TAB "   "

FILE* myOutput = stdout;
char* myOutputName;

void closeDumpFile ()
        {
        if (myOutput != stdout) fprintf (myOutput, "\n//DUMP file %s ends...", myOutputName);
        if (myOutput) fclose (myOutput);
```

```
        myOutput = stdout;
        };

int setDumpFile (char* name)
        {
        if (!(myOutput = fopen(name, "w")))
                {
                myOutput = stdout;
                return 0;
                };
        myOutputName = name;
        fprintf (myOutput, "//DUMP file %s begins...", name);
        return 1;
        }

void putTab (int tab)
        { for (int i = 0; i < tab; i++) fprintf (myOutput, TAB); }

void dump (char c, int tab)             { fprintf (myOutput, "%c (\\x%02x) ", c, c); }
void dump (int i, int tab)              { fprintf (myOutput, "%d ", i); }
void dump (double d, int tab)           { fprintf (myOutput, "%lg ", d); }

void dump (const char* s, int tab)      { fprintf (myOutput, "'%s' ", s); }
void dump (const int* i, int tab)       { fprintf (myOutput, "%d ",  *i); }
void dump (const double* d, int tab)    { fprintf (myOutput, "%lg ", *d); }

void dump (CObj* o, int tab)            { o->dump (tab); }
void dump (CObj& o, int tab)            { o.dump (tab); }

//---------------------------------------------------------------------------
```

Файл MsgSend\IPXDUMP.CPP

```
//-------------------------------------------------------------
//ipxdump.cpp, MSGSend ver 0.03, (c) Sunny, 2002-2003
//-------------------------------------------------------------

/*
dumplevels:
0 - no dump
1 - packet information
2 - packet dump
3 - ipx send/recieve
4 - ESRs, flags, timeouts - protocol states
5 - debug dump
*/

#include <time.h>
#include <stdio.h>
#include <string.h>


int dumplevel [6] = {0};
int dumppacketlen = 15;
char* modulename = "MSGSend";
FILE* ipxdump = stdout;

void openIpxDump (const char* name)
        {
        if (!(ipxdump = fopen (name, "w")))
                printf ("MSGSend ERROR 02: Cannot open dump file %s", name);
        else
                {
                time_t t = time (NULL);
                char* timestr = ctime (&t);
                int len = strlen (timestr);
                timestr[len-1] = 0x00;
                fprintf (ipxdump, "//MSGSend dump file %s begins at %s\n", name, timestr);
                }
        }
void closeIpxDump ()
        {
        if (ipxdump != stdout)
                {
                time_t t = time (NULL);
                char* timestr = ctime (&t);
                int len = strlen (timestr);
                timestr[len-1] = 0x00;
```

```
                fprintf (ipxdump, "//MSGSend dump file ends at %s\n", timestr);
                }
        fclose (ipxdump);
        }

int checkDumpLevel (int dl)
        { return dumplevel[dl]; }

void dumpHead (int dumplevel)
        {
        fprintf (ipxdump, "%i ", dumplevel);

        struct tm *time_now;
        time_t secs_now;
        char str[15];
        time (&secs_now);
        time_now = localtime (&secs_now);
        strftime (str, 15, "%y%m%d%I%M%S", time_now);
        fprintf (ipxdump, "%s: ", str);

        fprintf (ipxdump, "%s: ", modulename);
        }

#define NODEADDRLEN 6
void dumpNode (int num, char* adr)
        {
        switch (num)
        {
        case 0:
                {
                fprintf (ipxdump, "Node ??   [", num);
                for (int i = 0; i < NODEADDRLEN; i++)
                        fprintf(ipxdump, "%02X", (unsigned char)adr[i]);
                break;
                }
        case -1:
                {
                fprintf (ipxdump, "Broadcast [", num);
                for (int i = 0; i < NODEADDRLEN; i++)
                        fprintf(ipxdump, "%02X", (unsigned char)0xFF);
                break;
                };
        default:
                {
                fprintf (ipxdump, "Node %02i   [", num);
                for (int i = 0; i < NODEADDRLEN; i++)
                        fprintf(ipxdump, "%02X", (unsigned char)adr[i]);
                break;
                }
        }
        fprintf (ipxdump, "]: ");
        }

void dumpPacket (char* pack)
        {
        for (int i = 0; i < dumppacketlen; i++)
                fprintf (ipxdump, "%02X ", (unsigned char)pack[i]);
        fprintf (ipxdump, "\n");
        }
```

Файл MsgSend\ESRS.CPP

```
//-------------------------------------------------------------
//esrs.cpp, MSGSend ver 0.03, (c) Sunny, 2002-2003
//-------------------------------------------------------------

#define TIMEOUTEXIT 100
#define WAITPERIOD 2
#define EXITKEY 27 //esc

#include "cobj.cpp"
#include "clist.cpp"
#include "cnet.cpp"
#include "cinfo.cpp"
#include "ipxdump.cpp"
#include "esr.cpp"
#include "checkrs.cpp"
```

```
void checkR (CInfo& info, CNet& net)
        { check_code_send (info, net); }

void sendSAP ()
        {
        ECB ecbSafe = ecbS;
        IPXHEADER ipxHeaderSafe = ipxHeaderS;
        char bufferSafe [BUFFERLEN] = {0};
        for (int i = 0; i < BUFFERLEN; i++)     bufferSafe[i] = bufferS[i];

        char buf [10] = {0};
        ipxGetInternetworkAddress ((INTNETADDR*) buf);
        bufferS[0] = 0x00;
        for (i = 0; i < NODEADDRLEN; i++) bufferS[1 + i] = buf [4 + i];

        for (i = 0; i <  NETADDRLEN; i++) ipxHeaderS.destAddr.intAddr.network.data[i] = 0x00;
        for (i = 0; i < NODEADDRLEN; i++) ipxHeaderS.destAddr.intAddr.node.data[i] = 0xFF;
        ipxHeaderS.destAddr.socket = SOCKETNUM;

        LOCALTARGET localTarget = {0};
        localTarget.intAddr = ipxHeaderS.destAddr.intAddr;
        if ((errno = ipxGetLocalTarget (&localTarget)) != IPX_SUCCESS)
                printf ("ipxGetLocalTarget() error 0x%02X (%s)\n", errno, ipxErrorMsg (errno));

        ecbS.socket             = SOCKETNUM;
        ecbS.immediateAddr      = localTarget.immediateAddr;
        ecbS.fragmentCount      = 2;
        ecbS.fragment[0].buffer = &ipxHeaderS;
        ecbS.fragment[0].length = sizeof (ipxHeaderS);
        ecbS.fragment[1].buffer = bufferS;
        ecbS.fragment[1].length = sizeof (bufferS);

        ipxSendPacket (&ecbS);
        ipxRelinquishControl ();

        if (checkDumpLevel (1))
                {
                dumpHead (1); dumpNode (-1, NULL);
                fprintf (ipxdump, "Service advertising\n");
                }

        ecbS = ecbSafe;
        ipxHeaderS = ipxHeaderSafe;
        for (i = 0; i < BUFFERLEN; i++) bufferS[i] = bufferSafe[i];
        }

int checkKbd ()
        {
        if (kbhit ()) if (getch () == EXITKEY) return 1;
        return 0;
        }

void checkCmd (int argc, const char* argv[])
        {
        if (argc < 2)
                {
                printf ("Syntax: MSGSEND infofile [dump options] [dumpfile]\n");
                printf ("Dump options:\t/1 - packet information\n");
                printf ("\t\t/2 - packet dump\n\t\t/3 - IPXAPI dump\n");
                printf ("\t\t/4 - protocol states\n\t\t/5 - debug dump\n");
                exit (0);
                }
        if (argv[1][0] == '/')
                {
                printf ("MSGSend ERROR 03: No info file\n");
                exit (1);
                }
        if (argc > 2)
                {
                for (int i = 2; i < argc; i++)
                        if (argv[i][0] == '/')
                                if (argv[i][1] >= '1' && argv[i][1] <= '5') dumplevel[argv[i][1] - '0'] =
1;
                if (argv[argc-1][0] != '/') openIpxDump (argv[argc-1]);
                }
        }

int main (int argc, const char* argv[])
        {
```

```
        printf ("MSGSend (c) Sunny, 2002-2003, ver 0.03\n");
        checkCmd (argc, argv);
        CInfo info;
        CNet net;
        if (!info.readFile (argv[1]))
                {
                printf ("MSGSend ERROR 01: Cannot open info file %s", argv[1]);
                closeIpxDump ();
                exit (0);
                }

        printf ("Initialization. ");
        if (!initIPX ()) exit (1);

        unsigned time = ipxGetIntervalMarker ();

        int waiting = 0;
        recieve ();
        while (waiting != WAITPERIOD)
                {
                if (flagR)
                        {
                        printf ("MSGSend ERROR 04: MSGSend is already working in the net
now\n");
                        exit (1);
                        };
                if (checkTimeout (time))
                        {
                        waiting++;
                        time = ipxGetIntervalMarker ();
                        }
                };

        printf ("Success...\n");
        sendSAP ();
        char buf [10] = {0};
        ipxGetInternetworkAddress ((INTNETADDR*) buf);
        bufferS[0] = 0x00;
        for (int i = 0; i < NODEADDRLEN; i++) bufferS[1 + i] = buf [4 + i];

        while (!checkKbd ())
                {
                if (checkTimeout (time))
                        {
                        if (checkDumpLevel (4))
                                {
                                dumpHead (4);
                                fprintf (ipxdump, "Protocol state: ");
                                fprintf (ipxdump, "Timeout\n");
                                }
                        sendSAP ();
                        cancelAll ();
                        flagR = flagS = 0;
                        send ();
                        time = ipxGetIntervalMarker ();
                        }
                else
                        {
                        if (flagS)
                                {
                                if (checkDumpLevel (4))
                                        {
                                        dumpHead (4);
                                        fprintf (ipxdump, "Protocol state: ");
                                        fprintf (ipxdump, "Send ESR executed\n");
                                        dumpHead (4);
                                        fprintf (ipxdump, "Protocol state: ");
                                        fprintf (ipxdump, "Send flag is not 0\n");
                                        }
                                flagS = 0;
                                recieve ();
                                };
                        if (flagR)
                                {
                                if (checkDumpLevel (4))
                                        {
                                        dumpHead (4);
                                        fprintf (ipxdump, "Protocol state: ");
                                        fprintf (ipxdump, "Recieve ESR executed\n");
```

```
                                        dumpHead (4);
                                        fprintf (ipxdump, "Protocol state: ");
                                        fprintf (ipxdump, "Recieve flag is not 0\n");
                                        }
                                flagR = 0;
                                checkR (info, net);
                                send ();
                                time = ipxGetIntervalMarker ();
                                };
                        };
                };
        closeIpxDump ();
        };
```

Файл MsgSend\CHECKRS.CPP

```
//-------------------------------------------------------------
//checkrs.cpp, MSGSend ver 0.03, (c) Sunny, 2002-2003
//-------------------------------------------------------------

/*
;-------------------------------------------------------------
;from sender    - 00 (1 byte), <addr   (6 bytes)> - SAP - addres of sender
;               - 01 (1 byte),  <length (1 byte)>, <text(length bytes)> - next text line
;               - 02 (1 byte), <number (2 bytes)> - ID number of THIS node
;from reciver   - 03 (1 byte), <number (2 bytes)> - asking for next text line
;               - 01 (1 byte), <addr   (6 bytes)> - new node with its addr
;               - 02 (1 byte), <number (2 bytes)> - kill node
;-------------------------------------------------------------
*/

#define SOCKETNUM 0x0420
#define NODEADDRLEN 6
#define NETADDRLEN 4

writeAddr (CNet& net, int num)
        {
        for (int i = 0; i < NODEADDRLEN; i++) destadr[i] = net.getNodeAdr(num)[i];
        }

void fromshow_03 (CInfo& info, CNet& net)
        {
        if (checkDumpLevel (2))
                {
                dumpHead (2); dumpNode (*(int*)(&bufferR[1]),
net.getNodeAdr(*(int*)(&bufferR[1])));
                fprintf (ipxdump, "Packet code %02i\n", (int)bufferR[0]);
                fprintf (ipxdump, "packetR: "); dumpPacket (bufferR);
                }

        int num = *(int*)(&bufferR[1]);
        writeAddr (net, num);
        int numRec = net.getCurRecord (num);
        clrAll ();
        bufferS[0] = 0x01;
        bufferS[1] = info[numRec].myLen;
        for (int i = 0; i < info[numRec].myLen; i++)
                bufferS[2 + i] = info[numRec].myText[i];
        bufferS[2 + i] = 0x00;
        if (info.getNumberOfRecs() == numRec) numRec = 1;
                else numRec++;
        net.setCurRecord (num, numRec);

        if (checkDumpLevel (2))
                {
                fprintf (ipxdump, "packetS: "); dumpPacket (bufferS);
                }

        if (checkDumpLevel (1))
                {
                dumpHead (1); dumpNode (num, net.getNodeAdr(num));
                fprintf (ipxdump, "Message sent, length %i\n", (int)bufferS[1]);
                fprintf (ipxdump, "%.20s...\n", &bufferS[2]);
                }
        }

void fromshow_01 (CInfo& info, CNet& net)
        {
```

```
        if (checkDumpLevel (2))
                {
                dumpHead (2); dumpNode (0, bufferR + 1);
                fprintf (ipxdump, "Packet code %02i\n", (int)bufferR[0]);
                fprintf (ipxdump, "packetR: "); dumpPacket (bufferR);
                }

        int num = net.addNode (bufferR + 1);
        writeAddr (net, num);
        clrAll ();
        bufferS[0] = 0x02;
        bufferS[1] = *(unsigned char*)(&num);

        if (checkDumpLevel (2))
                {
                fprintf (ipxdump, "packetS: "); dumpPacket (bufferS);
                }

        if (checkDumpLevel (1))
                {
                dumpHead (1); dumpNode (0, net.getNodeAdr(num));
                fprintf (ipxdump, "Connected as Node %02i\n", num);
                }
        }

void fromshow_02 (CInfo& info, CNet& net)
        {
        if (checkDumpLevel (2))
                {
                dumpHead (2); dumpNode (0, bufferR + 1);
                fprintf (ipxdump, "Packet code %02i\n", (int)bufferR[0]);
                fprintf (ipxdump, "packetR: "); dumpPacket (bufferR);
                }

        if (checkDumpLevel (1))
                {
                dumpHead (1); dumpNode (*(int*)(&bufferR[1]),
net.getNodeAdr(*(int*)(&bufferR[1])));
                fprintf (ipxdump, "Disconnected\n");
                }

        int num = *(int*)(&bufferR[1]);
        writeAddr (net, num);
        net.delNode (num);
        clrAll ();
        }

void check_code_send (CInfo& info, CNet& net)
        {
        switch (bufferR[0])
                {
                case 0x03: fromshow_03 (info, net); break;
                case 0x01: fromshow_01 (info, net); break;
                case 0x02: fromshow_02 (info, net); break;
                };
        }
```

ДАВЫДОВА МАРИЯ, ЛИЦЕЙ №1580

ЗДРАВСТВУЙТЕ, меня зовут Давыдова Мария, лицей 1580, разрешите **ПРЕДСТАВИТЬ** вам работу «**СИСТЕМА АВТОМАТИЗИРОВАННОГО ОПОВЕЩЕНИЯ ПОЛЬЗОВАТЕЛЕЙ В ЛОКАЛЬНОЙ СЕТИ**».

Слайд 1

Основной **ЗАДАЧЕЙ** системы является **ИНФОРМИРОВАНИЕ** пользователей небольшой локальной сети, например компьютерного класса, без прерывания их текущей работы. Так называемое «информирование **В ФОНОВОМ РЕЖИМЕ**».

Слайд 2

Система реализована в виде программного **КОМПЛЕКСА** типа **КЛИЕНТ-СЕРВЕР**, состоящего из отдельных **КОМПОНЕНТОВ**, взаимодействующих между собой с помощью **ВНУТРЕННЕГО ПРОТОКОЛА** VSMP (Very Simple Message Protocol). Эта архитектура **ПОЗВОЛЯЕТ** легко **ИЗМЕНЯТЬ** уже созданные компоненты и **ДОБАВЛЯТЬ** новые, для решения других задач.

Пока разработаны два компонента (клиент и сервер), а так же версия VSMP для решения данной задачи. Рассмотрим **КЛИЕНТСКУЮ ПРОГРАММУ**.

Слайд 3

В ее задачи входит **ПРИЕМ И ОТОБРАЖЕНИЕ** текстовых **СООБЩЕНИЙ** на экране. Это **РЕЗИДЕНТНАЯ** программа DOS, работающая **В МОДИФИЦИРОВАННОМ ВИДЕОРЕЖИМЕ** с 26ю текстовыми строками вместо стандартных 25и. Для плавного появления сообщения используется техника динамического **ПЕРЕОПРЕДЕЛЕНИЯ ЗНАКОГЕНЕРАТОРА**. Программа написана на языке ассемблера и имеет **МОДУЛЬНУЮ АРХИТЕКТУРУ**. Модули представляют собой библиотеки ассемблерных процедур, которые можно использовать повторно. Рассмотрим каждый из них.

ЗАГРУЗОЧНЫЙ МОДУЛЬ решает системные задачи, обеспечивает **РЕЗИДЕНТНУЮ РАБОТУ** клиента. В его функции входят: **ПЕРЕХВАТ** необходимых прерываний, работа с **ПАМЯТЬЮ** и контроль **ПОВТОРНЫХ ЗАПУСКОВ** клиента на данной рабочей станции. Для корректной работы программы необходим перехват системного прерывания таймера, клавиатуры, Video BIOS, DOS Safe, и мультиплексного прерывания DOS.

Модуль **РАБОТЫ СО ЗНАКОГЕНЕРАТОРОМ** занимается динамическим **ПЕРЕОПРЕДЕЛЕНИЕМ** символов и **РАЗМЕЩЕНИЕМ** строки с сообщением в нужном месте видеобуфера.

Модуль **РАБОТЫ С ВИДЕОАДАПТЕРОМ** обеспечивает отображение на экране **26ОЙ СТРОКИ** и своевременное

СОХРАНЕНИЕ/ВОССТАНОВЛЕНИЕ РЕГИСТРОВ CRT-контроллера. Теперь рассмотрим **СЕРВЕРНУЮ ЧАСТЬ**.

Слайд 4

В ее задачу входит **РЕГИСТРАЦИЯ** и ведение **СТАТИСТИКИ** клиентов, **РАСПРЕДЕЛЕНИЕ** сообщений и отображение текущего состояния комплекса, а так же всех событий происходящих в нем **(ДАМП)**. Программа также имеет **МОДУЛЬНУЮ АРХИТЕКТУРУ.** Для реализации поставленной задачи была разработана **БИБЛИОТЕКА КЛАССОВ** на языке C++.

В состав каждого из компонентов комплекса должен входить **МОДУЛЬ** поддержки протоколов.

Слайд 5

Протокола **IPX**, протокола **VSMP** и библиотеки **VSMP-OVER-IPX**. Поддержка **IPX** реализована в виде **БИБЛИОТЕК ФУНКЦИЙ** для удобного обращения к **ДРАЙВЕРУ** на языках C++ и ассемблера. Протокол **VSMP** – библиотека **ОБРАБОТЧИКОВ** пакетов. **VSMP-OVER-IPX** – **НАДСТРОЙКА** над IPX обеспечивающая необходимые **ТАЙМАУТЫ** и **АСИНХРОННУЮ РАБОТУ** с пакетами.

ТЕКУЩАЯ ВЕРСИЯ VSMP позволяет клиенту и серверу налаживать связь и обмениваться информацией. Существует стандартный **ФОРМАТ ПАКЕТА**, позволяющий расширять протокол без изменения основы. **ПРИНЯТОЕ** сообщение

ОТОБРАЖАЕТСЯ на **ДОПОЛНИТЕЛЬНОЙ** текстовой строке путем **ИЗМЕНЕНИЯ НАСТРОЕК** видеоадаптера, рассмотрим их.

Слайд 6

В составе стандартного **АДАПТЕРА VGA/SVGA** есть блок, отвечающий за управление электронно-лучевой трубкой монитора, называемый **CRT-КОНТРОЛЛЕРОМ**. Управление VGA на **АППАРАТНОМ** уровне осуществляется с помощью **ЧТЕНИЯ/ЗАПИСИ** соответствующих **РЕГИСТРОВ** адаптера. С помощью незначительных изменений внесенных в установки CRT-контроллера можно добиться отображение 26ой текстовой строки. **ПЛАВНОСТЬ** вывода текста достигается с помощью **ПЕРЕОПРЕДЕЛЕНИЯ ЗНАКОГЕНЕРАТОРА**.

Слайд 7

Дело в том что, каждый символ в текстовом режиме **ОПРЕДЕЛЯЕТСЯ БИТОВОЙ МАТРИЦЕЙ** 8*16. Побитовый сдвиг матрицы с определенным шагом создает эффект «выезжания» символа на знакоместо.

Слайд 8

Итак, при выполнении данной работы, достигнуты **СЛЕДУЮЩИЕ РЕЗУЛЬТАТЫ**: разработан **МОДУЛЬНЫЙ МАСШТАБИРУЕМЫЙ** программный **КОМПЛЕКС** типа клиент-сервер, клиентская программа работает в **МОДИФИЦИРОВАННОМ** текстовом режиме на

ДОПОЛНИТЕЛЬНОЙ строке, используя динамическое переопределение **ЗНАКОГЕНЕРАТОРА**, разработан так же открытый **ПРОТОКОЛ VSMP**, и библиотеки **ПОДДЕРЖКИ** протокола **IPX**.

Возможна так же **ДАЛЬНЕЙШАЯ РАБОТА** в следующих направлениях: разработка серверной части под ОС **NOVELL,** разработка клиентской части под ОС **WINDOWS** и поддержка протокола **TCP/IP,** разработка компонента и версии протокола для обмена **ЛИЧНЫМИ СООБЩЕНИЯМИ** между пользователя

Демонстрация

Теперь перейдем к **ДЕМОНСТРАЦИИ** комплекса. Для эмуляции локальной сети на одном компьютере используется **VIRTUAL PC 5.1.** На каждой виртуальной машине уже установлен драйвер IPX.

<ЗАПУСТИТЬ СЕРВЕР БЕЗ ПАРАМЕТРОВ>

СЕРВЕР запускается с **ПАРАМЕТРАМИ** командной строки, первым из которых является имя **ФАЙЛА С СООБЩЕНИЯМИ** для рассылки в сети.

<ОТКРЫТЬ ТЕКСТОВЫЙ ФАЙЛ>

ВТОРЫМ параметром являются разрешенные **УРОВНИ ДАМПА.** Включаем дамп пакетов и основных событий. Третий параметр это имя файла, в который записывается дамп, он не указывается, следовательно, дамп будет производиться **НА ЭКРАН.**

<ЗАПУСТИТЬ СЕРВЕР В ПАРАМЕТРАМИ INFOFILE \1 \2 >

Теперь запустим **КЛИЕНТ**. Сейчас он работает резидентно.

<ЗАПУСТИТЬ КЛИЕНТ, ПРОДЕМОНСТРИРОВАТЬ ЭКРАН ПАМЯТИ, ЗАПУСТИТЬ И ЗАВЕРШИТЬ ВС >

Можно увеличивать и уменьшать **СКОРОСТЬ** появления текста комбинациями клавиш ctrl-alt-left/ctrl-alt-right.

ПОВТОРНЫЙ ЗАПУСК клиента вызовет ошибку.

<ИЗМЕНИТЬ СКОРОСТЬ, ЗАПУСТИТЬ КЛИЕНТ ЕЩЕ РАЗ>

Сейчас продемонстрируем **ОДНОВРЕМЕННУЮ** работу сервера с более чем одним **КЛИЕНТОМ**.

<ЗАПУСТИТЬ КЛИЕНТ НА ТРЕТЬЕЙ МАШИНЕ>

КЛИЕНТ поддерживает некоторые **ОПЦИИ** командной строки. Краткая справка по опциям и горячим клавишам, номер клиента в списке сервера, количество оперативной памяти, которое занимает резидентная часть клиента.

<ПРОДЕМОНСТРИРОВАТЬ ОПЦИИ>

К сожалению, Virtual PC не эмулирует **ЗНАКОГЕНЕРАТОР**. Для демонстрации его динамического переопределения воспользуемся **ЭМУЛЯЦИЕЙ DOS**.

<ЗАПУСК ВТОРОЙ ВЕРСИИ КЛИЕНТА ИЗ-ПОД ЭМУЛЯЦИИ>

ВЕРНЕМСЯ к Virtual PC. Как видно, **КЛИЕНТЫ** до сих пор **РАБОТАЮТ**. **ЗАВЕРШИМ** работу одного из них.

<ЗАВЕРШИТЬ РАБОТУ ОДНОГО КЛИЕНТА, ПРОДЕМОНСТРИРОВАТЬ ДАМП>

СПАСИБО ЗА ВНИМАНИЕ. Сейчас я готова **ОТВЕТИТЬ** на все **ВОПРОСЫ**.