Московский государственный технический университет им. Н.Э.Баумана

Утверждаю:

Галкин В.А.

"__"_____2017 г.

Курсовая работа по дисциплине «Сетевые технологии в АСОИУ» «Локальная безадаптерная сеть»

Расчетно-пояснительная записка
(вид документа)

писчая бумага
(вид носителя)

14
(количество листов)

ИСПОЛНИТЕЛИ:

студенты группы ИУ5-64
Матвейчук И.А. "___"_____2017 г.
Журавлева У.В. "___"_____2017 г.
Повираева М.Л. "___"_____2017 г.

Москва - 2017

Содержание

Введение

Данная программа, выполненная в рамках курсовой работы по предмету «Сетевые технологии», предназначена для организации обмена текстовыми сообщениями между тремя соединёнными с помощью интерфейса RS232C компьютерами. Программа позволяет обмениваться компьютерам, соединенным через COM-порты, текстовыми сообщениями и делать широковещательную рассылку.

Требования к программе

К программе предъявляются следующие требования. Программа должна:

1. устанавливать соединение между компьютерами и контролировать его целостность;
2. обеспечивать правильность передачи и приема данных с помощью алгоритма кодирования сообщения циклическим кодом [15, 11];
3. обеспечивать функцию передачи сообщений.
4. обеспечивать функцию широковещательной отправки сообщений.

Программа должна выполняться под управлением OS Windows 7 или выше. Поэтому было решено выполнить реализацию программы с помощью среды разработки MS Visual Studio 2015.

Определение структуры программного продукта

При взаимодействии компьютеров между собой выделаются несколько уровней: нижний уровень должен обеспечивать соединение компьютера со средой передачи, а верхний – обеспечить интерфейс пользователя. Программа разбивается на три уровня: физический, канальный и прикладной (см. приложение «Структурная схема программы»).

- Физический уровень предназначен для сопряжения компьютера со средой передачи.
- Канальный уровень занимается установлением и поддержанием соединения, формированием и проверкой пакетов обмена протоколов верхних модулей.
- Прикладной уровень занимается выполнением задач программы.

Физический уровень

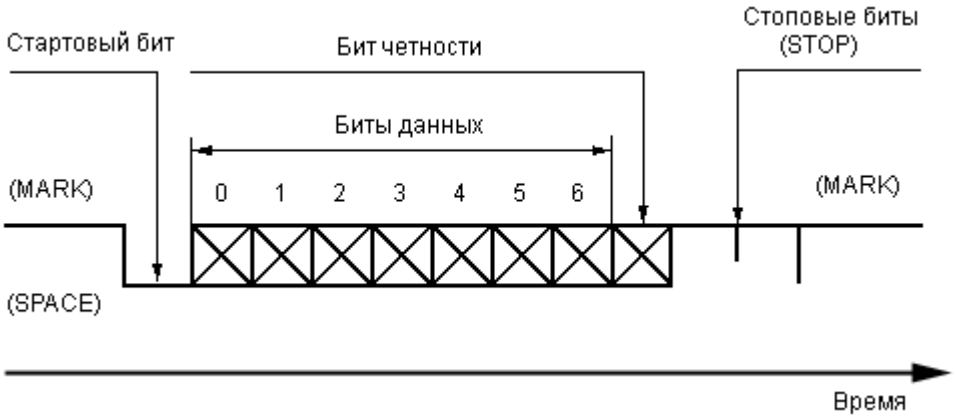
Функции физического уровня

Основными функциями физического уровня являются:

-установление параметров COM-порта,
-установление, поддержание и разъединение физического канала.

Описание физического уровня.

Последовательная передача данных означает, что данные передаются по единственной линии. При этом биты байта данных передаются по очереди с использованием одного провода. Для синхронизации группе битов данных обычно предшествует специальный *стартовый бит*, после группы битов следуют *бит проверки на четность* и один или два *стоповых бита*. Иногда бит проверки на четность может отсутствовать.



Из рисунка видно, что исходное состояние линии последовательной передачи данных – уровень логической 1. Это состояние линии называют отмеченным – **MARK**. Когда начинается передача данных, уровень линии переходит в 0. Это состояние линии называют пустым – **SPACE**. Если линия находится в таком состоянии больше определенного времени, считается, что линия перешла в состояние разрыва связи – **BREAK.**

Стартовый бит **START** сигнализирует о начале передачи данных. Далее передаются биты данных, вначале младшие, затем старшие.

Контрольный бит формируется на основе правила, которое создается при настройке передающего и принимающего устройства. Контрольный бит может быть установлен с контролем на четность, нечетность, иметь постоянное значение 1 либо отсутствовать совсем.

Если используется бит четности **P**, то передается и он. Бит четности имеет такое значение, чтобы в пакете битов общее количество единиц (или нулей) было четно или нечетно, в зависимости от установки регистров порта. Этот бит служит для обнаружения ошибок, которые могут возникнуть при передаче данных из-за помех на линии. Приемное устройство заново вычисляет четность данных и сравнивает результат с принятым битом четности. Если четность не совпала, то считается, что данные переданы с ошибкой. Конечно, такой алгоритм не дает стопроцентной гарантии обнаружения ошибок. Так, если при передаче данных изменилось четное число битов, то четность сохраняется, и ошибка не будет обнаружена. Поэтому на практике применяют более сложные методы обнаружения ошибок.

В самом конце передаются один или два стоповых бита **STOP**, завершающих передачу байта. Затем до прихода следующего стартового бита линия снова переходит в состояние **MARK.**

Использование бита четности, стартовых и стоповых битов определяют формат передачи данных. Очевидно, что передатчик и приемник должны использовать один и тот же формат данных, иначе обмен будет невозможен.

Другая важная характеристика – скорость передачи данных. Она также должна быть одинаковой для передатчика и приемника.

Скорость передачи данных обычно измеряется в бодах.

Иногда используется другой термин – биты в секунду (bps). Здесь имеется в виду эффективная скорость передачи данных, без учета служебных битов.

Интерфейс RS232C описывает несимметричный интерфейс, работающий в режиме последовательного обмена двоичными данными. Интерфейс поддерживает как асинхронный, так и синхронный режимы работы.

Последовательная передача данных означает, что данные передаются по единственной линии. При этом биты байта данных передаются по очереди с использованием одного провода. Интерфейс называется несимметричным, если для всех цепей обмена интерфейса используется один общий возвратный провод – сигнальная «земля».

Интерфейс 9-ти контактный разъем.

Номер контакта	Обозначение	Назначение	Обозначение CCITT
1	DCD	Обнаружение несущей	109
2	RD	Принимаемые данные	104
3	TD	Отправляемые данные	103
4	DTR	Готовность терминала к работе	108/2
5	SG	Земля сигнала (схемная)	102
6	DSR	Готовность DCE	107
7	RTS	Запрос передачи	105
8	TD	Отправляемые данные	103
9	RI	Индикатор вызова	125

В интерфейсе реализован биполярный потенциальный код на линиях между DTE и DCE. Напряжения сигналов в цепях обмена симметричны по отношению к уровню сигнальной «земли» и составляют не менее +3В для двоичного нуля и не более -3В для двоичной единицы.

Каждый байт данных сопровождается специальными сигналами «старт» – стартовый бит и «стоп» – стоповый бит. Сигнал «старт» имеет продолжительность в один тактовый интервал, а сигнал «стоп» может длиться один, полтора или два такта.

При синхронной передаче данных через интерфейс передаются сигналы синхронизации, без которых компьютер не может правильно интерпретировать потенциальный код, поступающий по линии RD.

Нуль-модемный интерфейс.

Обмен сигналами между адаптером компьютера и модемом (или 2-м компьютером присоединенным к исходному посредством кабеля стандарта RS-232C) строится по стандартному сценарию, в котором каждый сигнал генерируется сторонами лишь после наступления определенных условий. Такая процедура обмена информацией называется запрос/ответным режимом, или “**рукопожатием**” (**handshaking**). Большинство из приведенных в таблице сигналов как раз и нужны для аппаратной реализации “рукопожатия” между адаптером и модемом.

Обмен сигналами между сторонами интерфейса **RS-232C** выглядит так:

1. компьютер после включения питания выставляет сигнал **DTR**, который постоянно удерживается активным. Если модем включен в электросеть и исправен, он отвечает компьютеру сигналом **DSR**. Этот сигнал служит подтверждением того, что **DTR** принят, и информирует компьютер о готовности модема к приему информации;
2. если компьютер получил сигнал **DSR** и хочет передать данные, он выставляет сигнал **RTS**;
3. если модем готов принимать данные, он отвечает сигналом **CTS**. Он служит для компьютера подтверждением того, что **RTS** получен модемом и модем готов принять данные от компьютера. С этого момента адаптер может бит за битом передавать информацию по линии **TD**;
4. получив байт данных, модем может сбросить свой сигнал **CTS**, информируя компьютер о необходимости “притормозить” передачу следующего байта, например, из-за переполнения внутреннего буфера; программа компьютера, обнаружив сброс **CTS**, прекращает передачу данных, ожидая повторного появления **CTS**.
5. Когда модему необходимо передать данные в компьютер, он (модем) выставляет сигнал на разъеме 8 – **DCD**. Программа компьютера, принимающая данные, обнаружив этот сигнал, читает приемный регистр, в который сдвиговый регистр “собрал” биты, принятые по линии приема данных **RD**. Когда для связи используются только приведенные в таблице данные, компьютер не может попросить модем “повременить” с передачей следующего байта. Как следствие, существует опасность переопределения помещенного ранее в приемном регистре байта данных вновь “собранным” байтом. Поэтому при приеме информации компьютер

должен очень быстро освобождать приемный регистр адаптера. В полном наборе сигналов **RS-232C** есть линии, которые могут аппаратно “приостановить” модем.

Нуль-модемный интерфейс характерен для прямой связи компьютеров на небольшом расстоянии (длина кабеля до 15 метров). Для нормальной работы двух непосредственно соединенных компьютеров нуль-модемный кабель должен выполнять следующие соединения:

1. RI-1 + DSR-1 — DTR-2;
2. DTR-1 — RI-2 + DSR-2;
3. CD-1 — CTS-2 + RTS-2;
4. CTS-1 + RTS-1 — CD-2;
5. RD-1 — TD-2;
6. TD-1 — RD-2;
7. SG-1 — SG-2;

Знак «+» обозначает соединение соответствующих контактов на одной стороне кабеля.

Настройка COM-порта средствами C#

Пространство имен System.IO.Ports предлагает широкие возможности по настройке COM-порта.

Описание класса SerialPort

Этот класс используется для управления файловым ресурсом последовательного порта. Данный класс предоставляет возможности управления вводом-выводом в синхронном режиме или на основе событий, доступа к состоянию линии и состоянию разрыва, а также доступа к свойствам последовательного драйвера.

Методы класса:

Имя	Описание
Close	Закрывает соединение порта, присваивает свойству IsOpen значение false и уничтожает внутренний объект Stream.
CreateObjRef	Создает объект, который содержит всю необходимую информацию для создания прокси-сервера, используемого для взаимодействия с удаленным объектом. (Унаследовано от MarshalByRefObject.)
DiscardInBuffer	Удаляет данные из буфера приема последовательного драйвера.
DiscardOutBuffer	Удаляет данные из буфера передачи последовательного драйвера.
Dispose()	Освобождает все ресурсы, используемые объектом Component. (Унаследовано от Component.)
Dispose(Boolean)	Освобождает неуправляемые ресурсы, используемые объектом SerialPort (при необходимости освобождает и управляемые ресурсы). (Переопределяет Component.Dispose(Boolean).)
Equals(Object)	Определяет, равен ли заданный объект Object текущему объекту Object. (Унаследовано от Object.)
Finalize	Освобождает неуправляемые ресурсы и выполняет другие операции очистки перед тем, как объект Component будет освобожден при сборке мусора. (Унаследовано от Component.)
GetHashCode	Играет роль хэш-функции для определенного типа. (Унаследовано от Object.)
GetLifetimeService	Извлекает объект обслуживания во время существования, который управляет политикой времени существования данного экземпляра. (Унаследовано от MarshalByRefObject.)
GetPortNames	Получает массив имен последовательных портов для текущего компьютера.
GetService	Возвращает объект, представляющий службу, обеспечиваемую

	компонентом Component или его контейнером Container. (Унаследовано от Component.)
GetType	Возвращает объект Type для текущего экземпляра. (Унаследовано от Object.)
InitializeLifetimeService	Возвращает объект обслуживания во время существования для управления политикой времени существования данного экземпляра. (Унаследовано от MarshalByRefObject.)
MemberwiseClone	Создает неполную копию текущего объекта Object. (Унаследовано от Object.)
MemberwiseClone(Boolean)	Создает неполную копию текущего объекта MarshalByRefObject. (Унаследовано от MarshalByRefObject.)
Open	Открывает новое соединение последовательного порта.
Read(Byte[], Int32, Int32)	Считывает из входного буфера SerialPort определенное число байтов и записывает их в байтовый массив, начиная с указанной позиции.
Read(Char[], Int32, Int32)	Считывает из входного буфера SerialPort определенное число символов и записывает их в символьный массив, начиная с указанной позиции.
ReadByte	Считывает из входного буфера SerialPort один байт в синхронном режиме.
ReadChar	Считывает из входного буфера SerialPort один символ в синхронном режиме.
ReadExisting	Считывает все непосредственно доступные байты в соответствии с кодировкой из потока и из входного буфера объекта SerialPort.
ReadLine	Считывает данные из входного буфера до значения NewLine.
ReadTo	Считывает из входного буфера строку до указанного значения value.
ToString	Возвращает строку String, содержащую имя компонента Component, если таковое имеется.Этот метод не следует переопределять. (Унаследовано от Component.)
Write(String)	Записывает указанную строку в последовательный порт.
Write(Byte[], Int32, Int32)	Записывает указанное число байтов в последовательный порт, используя данные из буфера.
Write(Char[], Int32, Int32)	Записывает указанное число символов в последовательный порт, используя данные из буфера.
WriteLine	Записывает указанную строку и значение NewLine в выходной буфер.

Свойства класса:

Имя	Описание
BaseStream	Получает базовый объект Stream для объекта SerialPort.
BaudRate	Получает или задает скорость передачи для последовательного порта (в бодах).
BreakState	Получает или задает состояние сигнала разрыва.
BytesToRead	Получает число байтов данных, находящихся в буфере приема.
BytesToWrite	Получает число байтов данных, находящихся в буфере отправки.
CanRaiseEvents	Возвращает значение, показывающее, может ли компонент вызывать событие. (Унаследовано от Component.)
CDHolding	Получает состояние линии обнаружения несущей для порта.
Container	Возвращает контейнер IContainer, содержащий компонент Component. (Унаследовано от Component.)
CtsHolding	Получает состояние линии готовности к приему.

DataBits	Получает или задает стандартное число битов данных в байте.
DesignMode	Возвращает значение, указывающее, находится ли данный компонент Component в режиме конструктора в настоящее время. (Унаследовано от Component.)
DiscardNull	Получает или задает значение, показывающее, игнорируются ли пустые байты (NULL), передаваемые между портом и буфером приема.
DsrHolding	Получает или задает состояние сигнала готовности данных (DSR).
DtrEnable	Получает или задает значение, включающее поддержку сигнала готовности терминала (DTR) в сеансе последовательной связи.
Encoding	Получает или задает кодировку байтов для преобразования текста до и после передачи.
Events	Возвращает список обработчиков событий, которые прикреплены к этому объекту Component. (Унаследовано от Component.)
Handshake	Получает или задает протокол установления связи для передачи данных через последовательный порт.
IsOpen	Получает значение, указывающее состояние объекта SerialPort — открыт или закрыт.
NewLine	Получает или задает значение, используемое для интерпретации окончания вызова методов ReadLine и WriteLine.
Parity	Получает или задает протокол контроля четности.
ParityReplace	Получает или задает байт, которым заменяются недопустимые байты потока данных при обнаружении ошибок четности.
PortName	Получает или задает последовательный порт, в частности, любой из доступных портов COM.
ReadBufferSize	Получает или задает размер входного буфера SerialPort.
ReadTimeout	Получает или задает срок ожидания в миллисекундах для завершения операции чтения.
ReceivedBytesThreshold	Получает или задает число байтов, содержащихся во внутреннем входном буфере перед наступлением события DataReceived.
RtsEnable	Получает или задает значение, показывающее, включен ли сигнал запроса передачи (RTS) в сеансе последовательной связи.
Site	Получает или задает экземпляр ISite для компонента Component. (Унаследовано от Component.)
StopBits	Получает или задает стандартное число стоповых битов в байте.
WriteBufferSize	Получает или задает размер выходного буфера последовательного порта.
WriteTimeout	Получает или задает срок ожидания в миллисекундах для завершения операции записи.

События класса:

Имя	Описание
DataReceived	Представляет метод обработки события получения данных для объекта SerialPort.
Disposed	Происходит при удалении компонента вызовом метода Dispose. (Унаследовано от Component.)
ErrorReceived	Представляет метод обработки события ошибки объекта SerialPort.
PinChanged	Представляет метод для обработки события изменения последовательной линии объекта SerialPort.

1.1. Описание класса CommunicationManager

Поля класса:

Имя	Описание
f2	Экземпляр класса Form2Form, для доступа к элементам формы
_phys_connect	Экземпляр класса PhysicalConnection
coder	Экземпляр класса CycleCode
Start	Стартовый бит, равный 0xFF
Users	Словарь для связи COM-порта с пользователем
Speeds	Словарь для сравнения скорости COM-портов
reSendCount	Количество попыток пересылки сообщения
previousOperation	Переменная для запоминая операции
last_msg	Переменная для запоминая сообщения
last_msg_stat	Переменная для запоминая статуса сообщения (личное или широковещательное)

Перечисления класса:

Имя	Описание
MessageType	Содержит типы сообщений.
FrameType	Содержит типы кадров

Методы класса:

Имя	Описание
FrameAnalysis	Анализирует типы входных кадров
WriteData	Записывает указанную строку и тип кадра в последовательный порт.
DisplayData	Отображает данные с последовательного порта на экране
CharToInt	Переводит символ типа char в его числовое представление
ShowUsers	Обновление списка активных пользователей
WriteHistory	Запись сообщений в файл истории

1.2. Описание класса CycleCode

2. *Поля класса:*

Имя	Описание
mistake	Флаг ошибки при декодировании

Методы класса:

Имя	Описание
Code	Кодирует строку типа string при помощи циклического кода
Decode	Декодирует набор байтов в строку типа string с помощью циклического кода
Code1	Переводит строку типа string в массив байтов
Decode1	Переводит массив байтов в строку типа string
division	Алгоритм деления двоичных векторов и получения остатка
conc	Алгоритм конкатенации двоичных векторов
CycleCoding	Кодирует двоичный 11-ти разрядный вектор циклическим кодом
CycleDecoding	Декодирует двоичный 11-ти разрядный вектор циклическим кодом и сообщает об ошибке
StrToByte	Перевод строки, состоящей из 0 и 1 в вектор байт

Канальный уровень.

Функции канального уровня.

На канальном уровне выполняются следующие функции:

-запрос логического соединения,
-управление передачей кадров,
-обеспечение необходимой последовательности блоков данных,
передаваемых через межуровневый интерфейс,
-контроль и исправление ошибок,
-запрос на разъединение логического соединения.

2.1. Протокол связи.

В основном протокол содержит набор соглашений или правил, которого должны придерживаться обе стороны связи для обеспечения получения и корректной интерпретации информации, передаваемой между двумя сторонами. Таким образом, помимо управления ошибками и потоком протокол связи регулирует также такие вопросы, как формат передаваемых данных — число битов на каждый элемент и тип используемой схемы кодирования, тип и порядок сообщений, подлежащих обмену для обеспечения (свободной от ошибок и дубликатов) передачи информации между двумя взаимодействующими сторонами.

Перед началом передачи данных требуется установить соединение между двумя сторонами, тем самым проверяется доступность приемного устройства и его готовность воспринимать данные. Для этого передающее устройство посылает специальную команду: запрос на соединение, сопровождаемую ответом приемного устройства, например о приеме или отклонении вызова.

Также необходимо информировать пользователя о неисправностях в физическом канале, поэтому для поддержания логического соединения необходимо предусмотреть специальный кадр, который непрерывно будет посылаться с одного компьютера на другой, сигнализируя тем самым, что логическое соединение активно.

Защита передаваемой информации.

При передаче данных по линиям могут возникать ошибки, вызванные электрическими помехами, связанными, например, с шумами, порожденными коммутирующими элементами сети. Эти помехи могут вызвать множество ошибок в цепочке последовательных битов.

Метод четности/нечетности контрольная сумма блока не обеспечивают надежного обнаружения нескольких (например, двух) ошибок. Для этих случаев чаще всего применяется альтернативный метод, основанный на полиномиальных кодах. Полиномиальные коды используются в схемах покадровой (или поблочной) передачи. Это означает, что для каждого передаваемого кадра формируется (вырабатывается) один-единственный набор контрольных разрядов, значения которых зависят от фактического содержания кадра и присоединяются передатчиком к “хвосту” кадра. Приемник выполняет те же вычисления с полным содержимым кадра; если при передаче ошибки не возникли, то в результате вычислений должен быть получен заранее известный ответ. Если этот ответ не совпадает с ожидаемым, то это указывает на наличие ошибок.

Опишем кратко математический аппарат кодирования алгоритмом Хемминга.

Алгоритм кодирования состоит в том, что каждый байт, подлежащий кодированию, дополняется нулями до 11 знаков, после чего делится на полином. Все закодированные байты объединяются в строку, а строка переводится в массив байтов. В итоге из каждого байта получается два. На принимающей стороне производится обратные операции, определяем частное и остаток. По остатку определяем вектор ошибки, если остаток нулевой, то данные дошли безошибочно и мы отсылаем положительную квитанцию, если же ненулевой, то отсылаем отрицательную квитанцию — просьбу повторить посылку пакета.

Этапы кодирования циклическим кодом

1. Умножить исходный кодовый полином $m(x) = m_{k-1}x^{k-1} + ... + m_1x + m_0$ на (x^{n-k}) , что соответствует сдвигу кодового вектора в сторону старших разрядов на (n-k) разрядов:

$$x^{n-k} \bullet m(x) = m_{k-1}x^{n-1} + ... + m_1x^{n-k+1} + m_0x^{n-k}$$

2. Получить остаток p(x) от деления $x^{n-k} \bullet m(x)$ на порождающий полином g(x):

$$x^{n-k} \bullet m(x) = q(x) \bullet g(x) + p(x),$$

где q(x) – частное, p(x) - остаток. Так как степень полинома g(x) равна (n-k), то степень остатка должна быть равна (n-k-1) или быть меньше:

$$p(x) = p_{n-k-1}x^{n-k-1} + ... + p_2x^2 + p_1x + p_0.$$

3. Так как левая часть уравнения $p(x) + xn\text{-}k \bullet m(x) = q(x) \bullet g(x)$ кратна g(x), её можно раскрыть в виде полинома степени (n-1):

$$x^{n-k} \bullet m(x) + p(x) = m_{k-1}x^{n-1} + ... + m_1x^{n-k+1} + m_0x^{n-k} + p_{n-k-1}x^{n-k-1} + ... + p_1x + p_0.$$

И поэтому можно выполнить операцию конкатенации полученного кодового вектора остатка p(x) и исходного кодового вектора полинома m(x): $m_{k-1}....m_1m_0p_{n-k-1}....p_2p_1p_0$.

Декодирование циклическим кодом

Пусть: v(x) - передаваемый кодовый полином, r(x) – принятый кодовый полином. Разделив r(x) на порождающий полином g(x), можно получить:

$$r(x) = g(x) \bullet q(x) + s(x),$$

где q(x) – частное, s(x) – остаток.

Если остаток равен нулю, то есть принятый кодовый вектор является кратным порождающему полиному и, следовательно, ошибки нет или она не обнаружена.

Если остаток не равен нулю, то принятый кодовый вектор не является кодовым полиномом, то есть содержит ошибку.

Таким образом, ненулевой остаток определяет наличие ошибки, т.е. является ее синдромом:

$$s(x) = s_{n-k-1}x^{n-k-1} + ... + s_1x + s_0.$$

Пусть полином вектора ошибки имеет вид:

$$e(x) = e_{n-1}x^{n-1} + ... + e_1x + e_0.$$

Тогда $r(x) = v(x) + e(x)$, а $e(x) = v(x) + q(x) \bullet g(x) + s(x)$.

Так как v(x) – кодовый полином, кратный g(x), т.е. $v(x) = m(x) \bullet g(x)$, значит:

$e(x) = [m(x) + q(x)] \bullet g(x) + s(x)$.

Отсюда видно, что синдром s(x) является остатком от деления полинома вектора ошибок e(x) на порождающий полином g(x). В данной программе функция декодирования заключается в том, чтобы оценить синдром ошибки s(x) и определить, есть ошибка или её нет.

Для исправления ошибки необходимо инвертировать разряд, в котором допущена ошибка.

Формат кадров.

Кадры, передаваемые с помощью функций канального уровня, имеют различное назначение. Выделены супервизорные и информационные кадры.

Служебные супервизорные кадры.

Эти кадры используются для передачи служебной информации и реализуют следующие функции канального уровня: установление и разъединение логического канала, подтверждение приема информационного кадра без ошибок, запрос на повторную передачу принятого с ошибкой кадра. Формат эти кадров:

StartByte	Type	StopByte
Флаг начала кадра	Тип супервизорного кадра	Флаг конца кадра

Супервизорные кадры передачи параметров.

Супервизорные кадры передачи параметров используются для синхронизации параметров COM-портов, как принимающего, так и отправляющего. Кадр данного типа формируется, когда на одном из компьютеров изменяются параметры. Формат эти кадров:

StartByte	Type	Length	Data	StopByte
Флаг начала кадра	Тип супервизорного кадра	Длина поля Data	Параметры COM-порта	Флаг конца кадра

Информационные кадры.

Информационные кадры применяются для передачи закодированных циклическим кодом пользовательских сообщений. Формат эти кадров:

StartByte	Type	Length	Data	StopByte
Флаг начала кадра	Тип супервизорного кадра	Длина поля Data	Закодированные данные (текстовая строка)	Флаг конца кадра

Кадр можно разделить на несколько блоков – флаг начала кадра, тип кадра, длина передаваемых данных, данные и флаг конца кадра.

Флаги начала и конца кадра представляют собой байты, с помощью которых программа выделяет кадр, определяя соответственно начало и конец кадра.

Поле типа кадра обеспечивает правильное определение и распознавание разновидностей кадров и обработки их соответствующими процедурами.

Данные представляют собой либо закодированную строку в информационном кадре или параметры порта в супервизорном кадре передачи параметров.

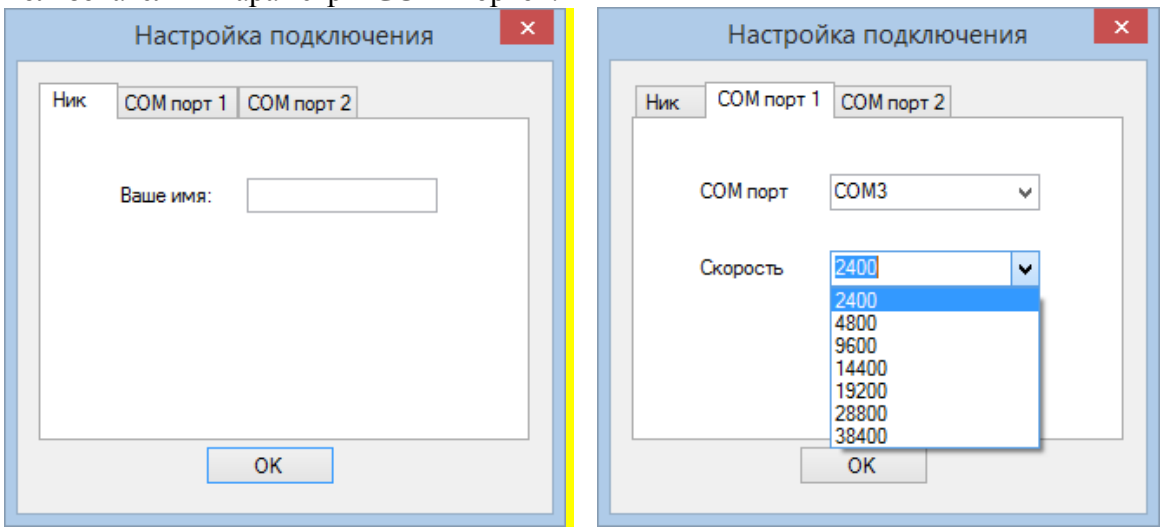
3. Прикладной уровень.

Функции прикладного уровня обеспечивают интерфейс программы с пользователем через систему форм и меню. Прикладной уровень предоставляет нижнему уровню текстовое сообщение.

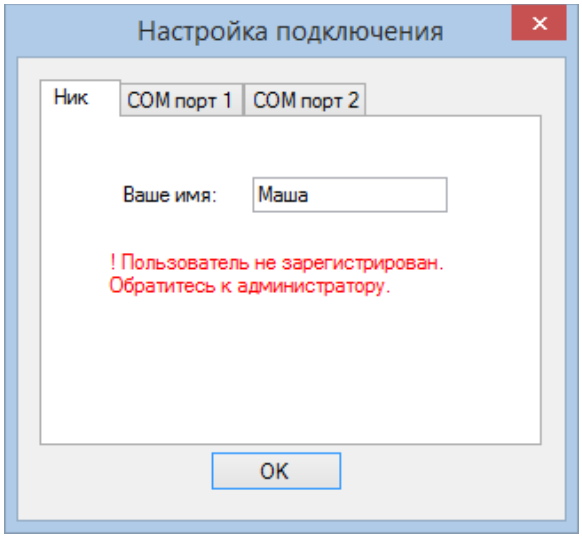
На данном уровне обеспечивается вывод принятых и отправленных сообщений в окно диалога пользователей.

Пользовательский интерфейс выполнен в среде Visual Studio 2015.

При запуске программы появляется форма, в которой необходимо ввести имя пользователя и параметры COM-портов.



После нажатия на клавишу “ОК” в том случае, если имя пользователя не внесено в список допустимых пользователей, будет выведено предупреждение и окно останется активным.



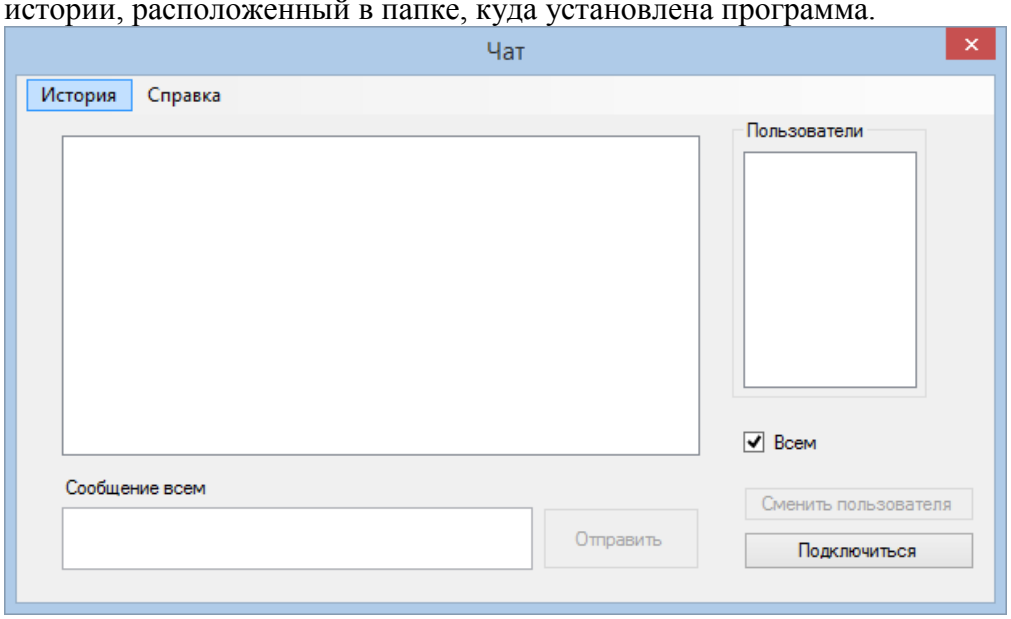
Если же пользователь зарегистрирован, окно закроется и откроется окно главной программы. При этом будет произведено открытие портов для дальнейшей работы.

Главным окном программы является окно «Чат». В данной форме есть следующие возможности:

1 Соединение с остальными компьютерами в сети.
2 Отправка сообщений
3 Отображение текущей истории
4 Смена пользователя
5 Выбор пользователя для отправки сообщения

6 Возможность выбора широковещательной передачи
7 Просмотр справки о программе

При помощи пункта меню «История» пользователь имеет возможность открыть файл истории, расположенный в папке, куда установлена программа.



При помощи пункта меню «Справка» пользователь может получить информацию о создателях программы.