



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ Информатика и системы управления _____

КАФЕДРА _____ Системы обработки информации и управления (ИУ5) _____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
по дисциплине «Сетевые технологии в АСОИУ»
НА ТЕМУ:

Программа пересылки диалога абонентов

Студент ИУ5Ц-81Б
(Группа)

(Подпись, дата) **Е.А. Коротенко**
(И.О.Фамилия)

Студент ИУ5Ц-82Б
(Группа)

(Подпись, дата) **Б.А. Пылаев**
(И.О.Фамилия)

Студент ИУ5Ц-82Б
(Группа)

(Подпись, дата) **А.Е. Чиварзин**
(И.О.Фамилия)

Руководитель курсовой работы

(Подпись, дата) **В.А. Галкин**
(В.А. Галкин)

Консультант

(Подпись, дата) **В.А. Галкин**
(И.О.Фамилия)

2022 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ5
(Индекс)

(И.О.Фамилия)
« ____ » февраля 20 22 г.

З А Д А Н И Е на выполнение курсовой работы

по дисциплине Сетевые технологии в АСОИУ

Студент группы _____

(Фамилия, имя, отчество)

Тема курсовой работы Программа пересылки диалога абонентов

Направленность КР (учебная, исследовательская, практическая, производственная, др.) _____

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения работы: 25% к 3 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

Задание Разработать протоколы взаимодействия объектов до прикладного уровня локальной сети, состоящей из 2-х ПК, соединённых нульмодемно через интерфейс RS232C, и реализующий функцию передачи текста диалога абонентов. Принимаемый и передаваемый тексты отображать в разных окнах. Скорость обмена и параметры СОМ-порта выбирает пользователь одного из ПК. Передаваемую информацию защитить [7,4] кодом Хэминга.

Оформление курсовой работы:

Расчетно-пояснительная записка на 57 листах формата А4.

Дата выдачи задания « 15 » февраля 2022 г.

Руководитель курсовой работы

(Подпись, дата)

В.А. Галкин
(И.О.Фамилия)

Студент

(Подпись, дата)

(И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

1. Введение

Данная программа, выполненная в рамках курсовой работы по предмету «Сетевые технологии в автоматизированных системах обработки информации и управления», предназначена для пересылки текста диалога абонентов между двумя соединёнными с помощью интерфейса RS232C компьютерами.

2. Требования к программе

Программное изделие выполняется на C# под управлением MS Windows.

Для работы программы требуются 2 ПК типа IBM PC AT (/XT), соединенные нульмодемным кабелем через интерфейс RS-232C.

Важно, чтобы пользователь обладал правами администратора в ОС Windows на ПК.

3. Требования к программе

См. Схему «Структурная схема программы»

4. Физический уровень

4.1 Сигналы интерфейса RS-232-C.

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*, как показано на рисунке 1. Иногда бит проверки на четность может отсутствовать.

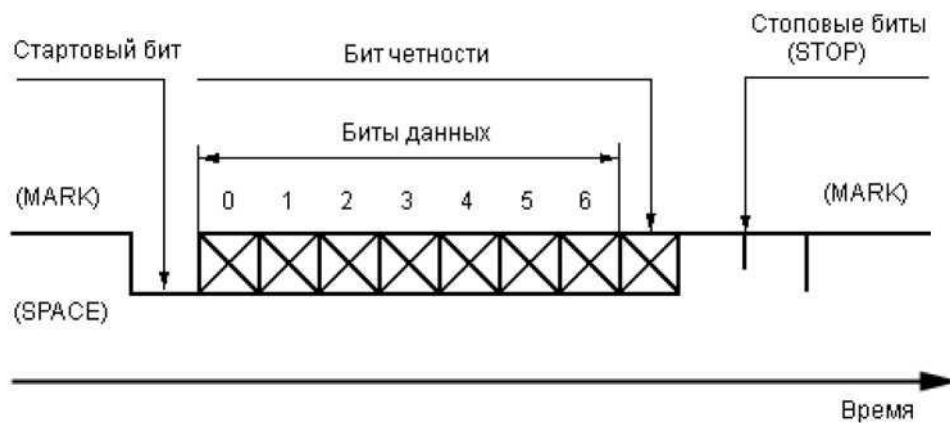


Рисунок 1.

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. Она также должна быть одинаковой для передатчика и приемника.

Скорость изменения информативного параметра сигнала 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.

Интерфейс называется несимметричным, если для всех цепей обмена интерфейса используется один общий возвратный провод — сигнальная «земля».

Интерфейсы 25-ти (DB25) или 9-ти (DB9) контактный разъем.

Наименование сигнала	Цепь	Номер контакта	
		DB25P	DB9S
DCD (Data Carrier Detect)	109	8	1
RD (Receive Data)	104	3	2
TD (Transmit Data)	103	2	3
DTR (Data Terminal Ready)	108	20	4
GND (Signal Ground)	102	7	5
DSR (Data Set Ready)	107	6	6
RTS (Request To Send)	105	4	7
CTS (Clear To Send)	106	5	8
RI (Ring Indicator)	125	22	9

Таблица 1.

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.

Входы TD и RD используются устройствами DTE и DCE по-разному. DTE использует вход TD для передачи данных, а вход RD для приема данных. И наоборот, устройство DCE использует вход TD для приема, а вход RD для передачи данных. Поэтому для соединения двух DTE необходимо перекрестное соединение линий TD и RD в нуль-модемном кабеле.

Рассмотрим самый низкий уровень управления связью - подтверждение связи.

В начале сеанса связи компьютер (DTE) должен удостовериться, что

модем (DCE) находится в рабочем состоянии. Для этой цели компьютер подает сигнал по линии DTR. В ответ модем подает сигнал по линии DSR. Затем, после вызова абонента, модем подает сигнал по линии DCD, чтобы сообщить компьютеру, что он произвел соединение с удаленной системой.

Более высокий уровень используется для управления потоком (скоростью обмена данными) и также реализуется аппаратно. Этот уровень необходим для того, чтобы предотвратить передачу большего числа данных, чем то, которое может быть обработано принимающей системой.

В полудуплексных соединениях DTE подает сигнал RTS, когда оно желает передать данные. DCE отвечает сигналом по линии CTS, когда оно готово, и DTE начинает передачу данных. До тех пор, пока оба сигнала RTS и CTS не примут активное состояние, только DCE может передавать данные. Иногда для соединения двух устройств DTE эти линии (RTS и CTS) соединяются вместе на каждом конце кабеля. В результате получаем то, что другое устройство всегда готово для получения данных (если при большой скорости передачи принимающее устройство не успевает принимать и обрабатывать данные, возможна потеря данных).

Для решения всех этих проблем для соединения двух устройств типа DTE используется специальный нуль-модемный кабель.

4.2 Нуль-модемный интерфейс

Обмен сигналами между адаптером компьютера (DTE) и модемом (DCE) (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 и открытия COM-порта выставляет сигнал **DTR**, который 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**.

Модем может передать данные в компьютер, когда он обнаружит несущую в линии и выставит сигнал — **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.

4.2.1. Настройка COM-порта средствами библиотеки для C#

Прикладная библиотека Serial COM для C# предлагает широкие возможности по настройке COM-порта. Подробное описание библиотеки

<https://docs.microsoft.com/ru-ru/dotnet/api/system.io.ports.serialport?view=netframework-4.8>

4.2.2. Описание класса SerialPort

Класс SerialPort дает возможность управления последовательными портами компьютера. Он определяет минимальную функциональность для работы с ними.

4.2.3. Конструкторы класса

<u>SerialPortQ</u>	Инициализация нового экземпляра класса <u>SerialPort</u> .
---------------------------	--

<u>SerialPort(IContainer)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанный объект <u>IContainer</u> .
<u>SerialPort(String)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта.
<u>SerialPort(String, Int32)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта и скорость передачи в бодах.
<u>SerialPort(String, Int32, Parity)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта, скорость передачи в бодах и бит четности.
<u>SerialPort(String, Int32, Parity, Int32)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта, скорость передачи в бодах, бит четности и биты данных.
<u>SerialPort(String, Int32, Parity, Int32, StopBits)</u>	Инициализирует новый экземпляр класса <u>SerialPort</u> , используя указанное имя порта, скорость передачи в бодах, бит четности, биты данных и стоп-бит.

Таблица 2.

4.2.4. Методы

<u>Close()</u>	Закрывает соединение порта, присваивает свойству <u>IsOpen</u> значение false и уничтожает внутренний объект <u>Stream</u> .
<u>CreateObjRef(Type)</u>	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object. (Унаследовано от <u>MarshalByRefObject</u>)
<u>DiscardInBuffer()</u>	Удаляет данные из буфера приема последовательного драйвера.
<u>DiscardOutBuffer()</u>	Удаляет данные из буфера передачи последовательного драйвера.
<u>Dispose()</u>	Освобождает все ресурсы, занятые <u>Component</u> . (Унаследовано от <u>Component</u>)
<u>Dispose(Boolean)</u>	Освобождает неуправляемые ресурсы, используемые <u>SerialPort</u> , и дополнительно освобождает управляемые ресурсы.
<u>Equals(Object)</u>	Определяет, равен ли указанный объект текущему объекту. (Унаследовано от <u>Object</u>)
<u>GetHashCode()</u>	Служит в качестве хэш-функции по умолчанию. (Унаследовано от <u>Object</u>)

<u>GetLifetimeService()</u>	Retrieves the current lifetime service object that controls the lifetime policy for this instance. (Унаследовано от <u>MarshalByRefObject</u>)
<u>GetPortNames()</u>	Получает массив имен последовательных портов для текущего компьютера.
<u>GetService(Type)</u>	Возвращает объект, представляющий службу, предоставляемую классом <u>Component</u> или классом <u>Container</u> . (Унаследовано от <u>Component</u>)
<u>GetType()</u>	Возвращает объект <u>Type</u> для текущего экземпляра. (Унаследовано от <u>Object</u>)
<u>InitializeLifetimeService()</u>	Obtains a lifetime service object to control the lifetime policy for this instance. (Унаследовано от <u>MarshalByRefObject</u>)
<u>MemberwiseClone()</u>	Создает неполную копию текущего объекта <u>Object</u> . (Унаследовано от <u>Object</u>)
<u>MemberwiseClone(Boolean)</u>	Creates a shallow copy of the current <u>MarshalByRefObject</u> object. (Унаследовано от <u>MarshalByRefObject</u>)
<u>Open()</u>	Открывает новое соединение последовательного порта.
<u>Read(Byte[], Int32, Int32)</u>	Считывает из входного буфера <u>SerialPort</u> определенное число байтов и записывает их в байтовый массив, начиная с указанной позиции.
<u>Read(Char[], Int32, Int32)</u>	Считывает из входного буфера <u>SerialPort</u> определенное число символов и записывает их в символьный массив, начиная с указанной позиции.
<u>ReadByte()</u>	Считывает из входного буфера <u>SerialPort</u> один байт в синхронном режиме.
<u>ReadChar()</u>	Считывает из входного буфера <u>SerialPort</u> один символ в синхронном режиме.
<u>ReadExisting()</u>	Считывает все непосредственно доступные байты в соответствии с кодировкой из потока и из входного буфера объекта <u>SerialPort</u> .
<u>ReadLine()</u>	Считывает данные из входного буфера до значения <u>NewLine</u> .
<u>ReadTo(String)</u>	Считывает из входного буфера строку до указанного значения value.

<u>ToString()</u>	Возвращает объект <u>String</u> , содержащий имя <u>Component</u> , если оно есть. Этот метод не следует переопределять. (Унаследовано от <u>Component</u>)
<u>Write(Byte[], Int32, Int32)</u>	Записывает указанное число байтов в последовательный порт, используя данные из буфера.
<u>Write(Char[], Int32, Int32)</u>	Записывает указанное число символов в последовательный порт, используя данные из буфера.
<u>Write(String)</u>	Записывает указанную строку в последовательный порт.
<u>WriteLine(String)</u>	Записывает указанную строку и значение <u>NewLine</u> в выходной буфер.

Таблица 3.

4.3. Функции физического уровня.

4.4.1. Открытие порта

В ОС Windows доступ к COM-портам предоставляется посредством файловых интерфейсов. В данной работе доступ предоставляется на основе 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.

Соответственно, для открытия порта в языке C# используется функция класса SerialPort Open(). В качестве параметров этому методу передаются параметры, настраивающие передачу данных на физическом уровне, упомянутые выше, а также имя доступного COM-порта для подключения. При успешном выполнении функции свойство IsOpen экземпляра класса SerialPort становится равным true, в противном случае генерируется одно из следующих исключений:

- UnauthorizedAccessException
- ArgumentOutOfRangeException

- ArgumentException
- IOException
- InvalidOperationException

Тайм-ауты необходимы для правильной работы функций чтения из порта и записи в порт. В программе определяются следующие значения:

- ReadTimeout определяет срок ожидания в миллисекундах для завершения операции чтения из порта.
- WriteTimeout определяет срок ожидания в миллисекундах для завершения операции записи в порт.

По истечении данных таймаутов в случае неуспешной записи/чтения генерируется исключение `TimeoutException` - задает сообщение с описанием источника исключения. Если метод вызывает это исключение, обычно выдается сообщение "Операция не завершена в связи с истечением тайм-аута".

Также при выполнении метода экземпляру `SerialPort` предоставляется базовый объект `Stream` для работы, - в целом, класс `Stream` является абстрактным базовым классом всех потоков. Поток — это абстракция последовательности байтов, например файл, устройство ввода-вывода, канал взаимодействия процессов или сокет TCP/IP. В данном случае `Stream` представляет собой абстракцию для канала, осуществляющего запись данных в порт.

4.4.2. Заккрытие порта

Заккрытие порта осуществляется путем вызова метода `Close()` на экземпляре класса `SerialPort`. Данный метод закрывает соединение порта, присваивает свойству IsOpen значение `false` и уничтожает внутренний объект Stream.

4.4.3. Передача данных

В данной работе передача данных между двумя компьютерами осуществляется вызова метода Write(Byte[], Int32, Int32) класса `SerialPort`.

Для передачи данных в классе `SerialPort` существует набор методов `Write`, отличающихся параметрами:

- Записывает указанную строку в последовательный порт.
- Записывает указанное число байтов в последовательный порт, используя данные из буфера.
- Записывает указанное число символов в последовательный порт, используя данные из буфера.

4.4.4. Прием данных

Прием данных реализуется как обработчик события `DataReceived` класса `SerialPort`, - данное событие активизируется при наличии данных во входном буфере порта. В рамках физического уровня обработчика данные считываются с помощью набора методов для считывания данных `Read`, отличающихся принимаемыми параметрами:

- Считывает из входного буфера `SerialPort` определенное число байтов и записывает их в байтовый 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`.

Для анализа числа поступивших байтов используется свойство `BytesToRead`, получающее число байтов данных, находящихся в буфере приема.

4.4.5. Поддержание соединения

Поддержание физического соединения осуществляется отправкой сигналов *LINKACTIVE* каждые 10 секунд и получением ответа *ACK_LINKACTIVE* о том, что соединение поддерживается.

5. Канальный уровень

5.1. Функции канального уровня

На канальном уровне должны выполняться следующие функции:

1. Запрос логического соединения;
2. Управление передачей кадров;
3. Обеспечение необходимой последовательности блоков данных, передаваемых через межуровневый интерфейс;
4. Контроль и обработка ошибок;
5. Проверка целостности логического соединения;
6. Посылка подтверждения.
7. Запрос на разъединение логического соединения.

5.2. Протокол связи

Важнейшими понятиями протокола связи являются управление ошибками и управление потоком. 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 следующие вопросы

- 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.

Например, 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 и ожидает ее приема с другого СОМ-порта.

5.3. 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 или полученная в развитие этого метода контрольная сумма блока не обеспечивают надежного обнаружения пачки 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 один-единственный набор контрольных разрядов, значения которых зависят от фактического содержания 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.

В данной программе передаваемая информация защищена кодом Хемминга [7,4].

5.4. Кодирование кодом Хемминга

Алгоритм кодирования состоит в том, что данные разбиваются на блоки фиксированной длины и ввода в них контрольных бит, дополняющих до четности несколько пересекающихся групп, охватывающих все биты блока.

Ричард Хемминг рассчитал минимальное количество проверочных бит, позволяющих однозначно исправлять однократные ошибки.

Если длина информационного блока, который требуется закодировать - m бит. Количество контрольных бит, используемых для его кодирования, - k , то закодированный блок будет иметь длину: $n = m + k$ бит. Для каждого блока такой длины возможны n различных комбинаций, содержащих ошибку.

Таким образом, для каждого передаваемого информационного блока может существовать n -блоков, содержащих однократную ошибку, и один блок - без ошибок. Следовательно, максимальное количество различных закодированных блоков, содержащих не больше одной ошибки, будет: $2^m(n + 1)$, где $n = m + k$.

Если для информационных данных длиной m подобрать такое количество контрольных бит k , что максимально возможное количество различных последовательностей длиной $m+k$ будет больше или равно максимальному количеству различных закодированных информационных блоков, содержащих не больше одной ошибки, то точно можно утверждать, что существует такой метод кодирования информационных данных с помощью k контрольных бит, который гарантирует исправление однократной ошибки.

Следовательно, минимальное количество контрольных бит, необходимых для исправления однократной ошибки, определяется из равенства:

$$2^m \times (n + 1) = 2^n$$

Учитывая, что $n = m + k$, получаем:

$$k = 2^k - m - 1$$

Так как количество бит должно быть целым числом, то k , вычисленное с помощью этого уравнения, необходимо округлить до ближайшего большего целого числа.

Например, для информационных данных длиной 7 необходимо 4 контрольных бита, чтобы обеспечить исправление однократных ошибок, а для данных длиной 128 бит необходимо 8 контрольных бит.

Мало определить минимальное количество контрольных бит, необходимых для исправления ошибки. Необходимо разработать алгоритм

проверки данных с помощью этих контрольных разрядов. Ричард Хемминг предложил следующий алгоритм.

Все биты, порядковые номера которых являются степенью двойки, – это контрольные разряды. То есть если порядковый номер бита обозначить символом 'n', то для контрольных бит должно быть справедливо равенство: $n = 2^k$, где k – любое положительное целое число.

Например, для закодированной последовательности длиной 13 бит проверочными будут: 1, 2, 4 и 8 биты, так как $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$.

Каждый выбранный, таким образом, контрольный бит будет проверять определенную группу бит, т.е. в контрольный бит будет записана сумма по модулю два всех битов группы (дополнение до четного количества единиц), которую он проверяет.

Для того, чтобы определить какими контрольными битами контролируют бит, необходимо разложить его порядковый номер по степени 2. Таким образом, девятый бит будет контролироваться битами 1 и 8, так как $9 = 2^0 + 2^3 = 1 + 8$.

5.5. Форматы кадров

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

5.1.1. Служебные супервизорные кадры.

UPLINK-кадр – кадр запроса на установление логического соединения

ACK-UPLINK-кадр – положительная квитанция на UPLINK-кадр

LINKACTIVE-кадр – кадр поддержания логического соединения

ACK-LINKACTIVE-кадр – положительная квитанция на LINKACTIVE-кадр

RET-кадр – кадр запроса повторной передачи сообщения при ошибке в сообщении (неправильность битов)

DOWNLINK-кадр – кадр разрыва логического соединения

ACK-DOWNLINK – положительная квитанция на DOWNLINK-кадр

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

5.1.2. 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	Data	StopByte
Флаг начала кадра	Тип супервизорного кадра	Параметры COM-порта	Флаг конца кадра

5.1.3. Информационные кадры.

DAT-кадр

Поскольку кадры имеют переменную длину, каждый поступающий кадр должен буферизоваться (т.е. сохраняться в памяти), что гарантирует его целостность до начала передачи.

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

StartByte	Type	Data
Флаг начала кадра	Тип супервизорного кадра	Закодированные данные (текстовая строка)

Кадр можно разделить на несколько блоков - флаг начала кадра, тип кадра, данные.

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.

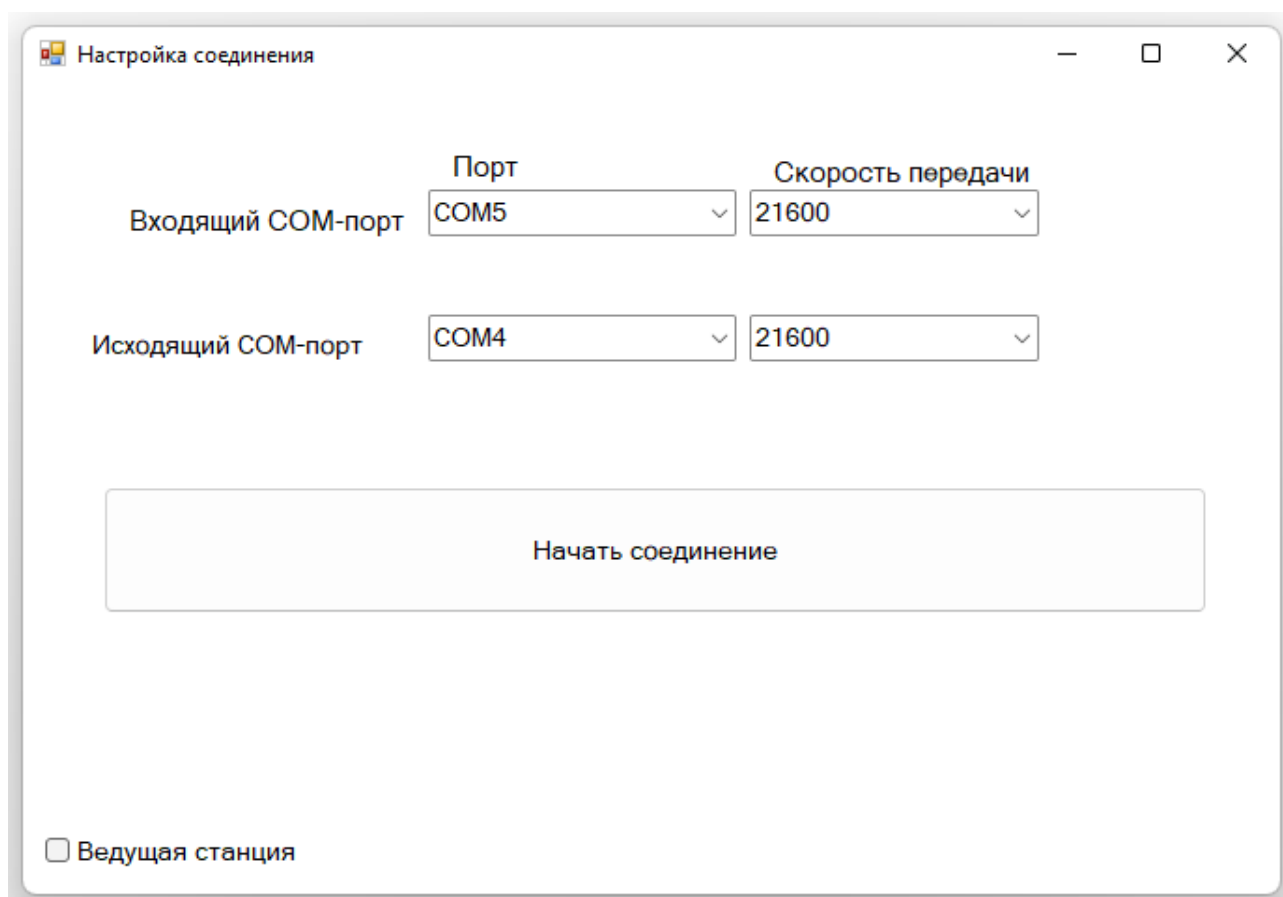
6. Прикладной уровень

Функции прикладного уровня обеспечивают интерфейс программы с пользователем.

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

Пользовательский интерфейс выполнен в среде Visual Studio 2022. При его разработке учитывались рекомендации по простоте, удобству и функциональности интерфейса.

При запуске программы появляется форма, в которой происходит настройка COM-портов компьютера (см рис. **Error! Reference source not found.**).



Настройка соединения

	Порт	Скорость передачи
Входящий COM-порт	COM5	21600
Исходящий COM-порт	COM4	21600

Начать соединение

☐ Ведущая станция

Рис. 2 Настройка COM-портов устройства

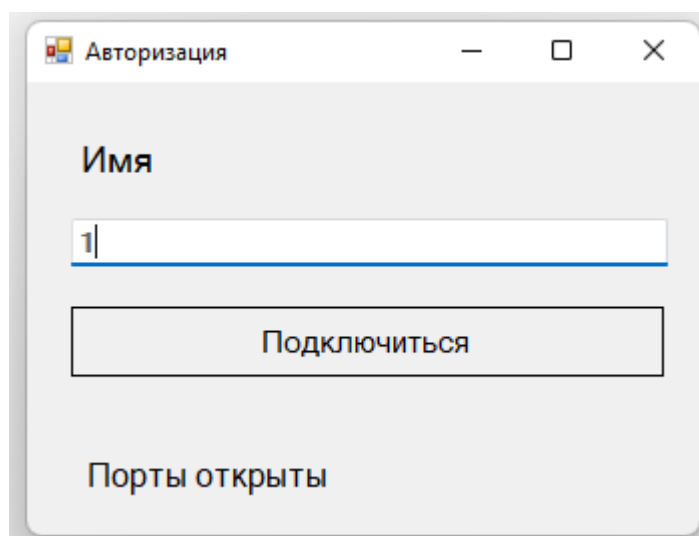


Рис. 3 Ввод имени пользователя

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

- 1 Подключение – создание соединения между устройствами
- 2 Отключение – разрыв соединения между устройствами
- 3 Отправка – позволяет отправить сообщение пользователю на другое устройство
- 4 Просмотр – в секции «Чат» пользователи могут наблюдать их совместную переписку

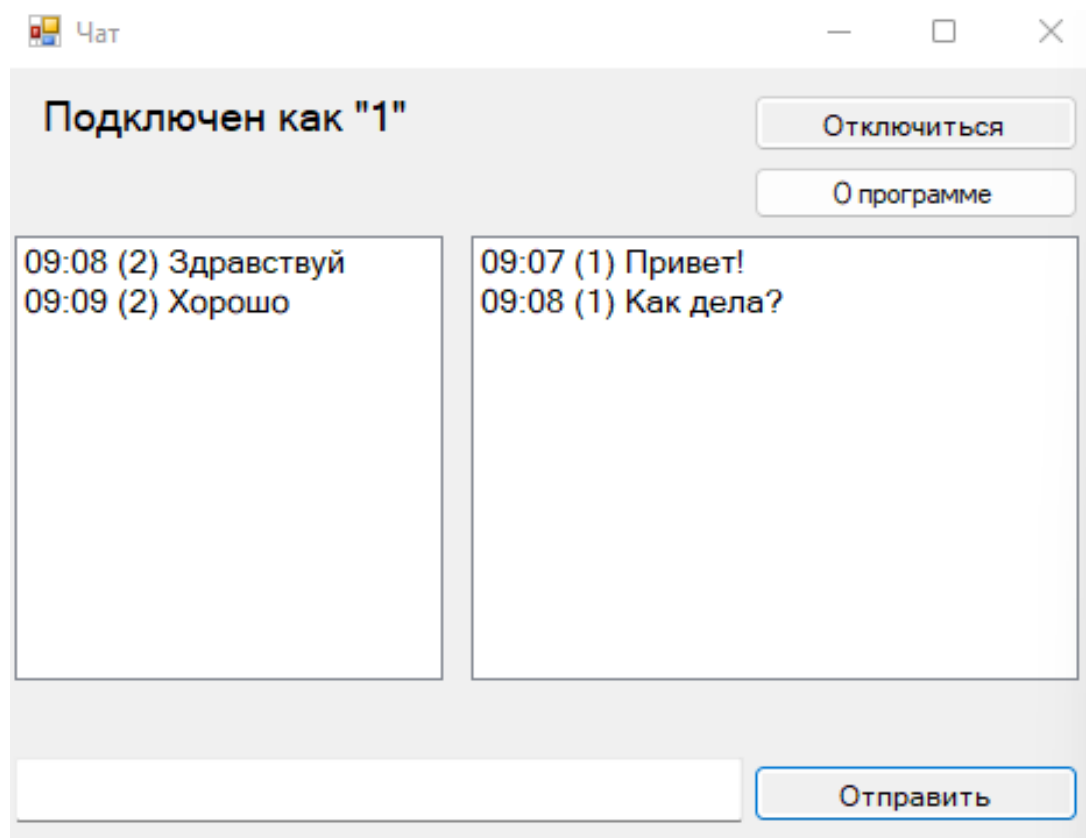


Рис. 4 Форма “Чат”

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1. Наименование:

Программа пересылки диалога абонентов.

2. Основание для разработки:

Основанием для разработки является учебный план
МГТУ им. Баумана кафедры ИУ5 на 8 семестр.

3. Исполнители:

Исполнителями являются студенты МГТУ им. Н.Э. Баумана группы ИУ5Ц82Б: Чиварзин А.Е. (пользовательский уровень), Пылаев Б.А. (канальный уровень), Коротенко Е.А. (физический уровень).

4. Цель разработки: разработать протоколы взаимодействия объектов до прикладного уровня локальной сети, состоящей из 2-х ПК, соединенных нульмодемно через интерфейс RS232C, и реализующей функцию передачи текста диалога абонентов. Принимаемый и передаваемый тексты отображать в разных окнах. Скорость обмена и параметры COM-порта выбирает пользователь одного из ПК. Передаваемую информацию защитить [7,4]-кодом Хэмминга.

5. Содержание работы:

5.1 Задачи, подлежащие решению:

- разработать протоколы взаимодействия объектов прикладного, канального и физического уровней локальной сети,
- защитить передаваемую информацию [7,4] кодом Хэмминга,
- реализовать функцию передачи текстовых сообщений,

5.2 Требования к программному изделию:

5.2.1 Требования к функциональным характеристикам:

Программа должна контролировать процессы, связанные с получением, использованием и освобождением различных ресурсов ПК. При возникновении ошибок обрабатывать их, а в случае необходимости:

- извещать пользователя своего ПК,
- извещать пользователя ПК на другом конце канала.

Номер COM-порта и параметры обмена устанавливается через меню.

5.2.2 На физическом уровне должны выполняться следующие функции:

- установление параметров COM-порта,

- установка, поддержание и разъединение физического канала.

5.2.3 На канальном уровне должны выполняться следующие функции:

- установка логического соединения,
- управление передачей кадров,
- обеспечение необходимой последовательности блоков данных, передаваемых через межузровеньный интерфейс,
- контроль и исправление ошибок,
- разрыв логического соединения.

5.2.4 На пользовательском уровне должны выполняться следующие функции:

- интерфейс с пользователем через систему меню,
- выбор режима работы,
- выбор номера СОМ-порта для канала,
- установка параметров СОМ-порта,
- передача параметров СОМ-порта на другой ПК.
- интерфейс для просмотра и отправки текстовых сообщений

5.3 Входные и выходные данные:

5.3.1 Входные данные:

Входными данными являются:

- текст сообщения, вводимый с клавиатуры передающего ПК.

5.3.2 Выходные данные:

- принятый текст сообщения на экране ПК.

6. Требования к составу технических средств:

Программное изделие выполняется на настольных ПК, на языке программирования C#, под управлением MS Windows.

Для демонстрации работы программы требуется 2 ПК, соединенных нульмодемным кабелем через интерфейс RS-232C. Допускается использование программного эмулятора нульмодемного соединения.

7. Этапы разработки:

7.1 Разработка Технического Задания до 15.02.2022г.

7.2 Разработка Эскизного Проекта до 25.02.2022г.

7.3 Разработка Технического Проекта до 30.03.2022г.

7.4 Разработка Программы до 30.04.2022г.

8. Техническая документация, предъявляемая по окончании работы:

8.1 Технический проект.

- Расчетно-пояснительная записка, включающая в приложении комплект технической документации на программный продукт, содержащий:

Приложение 1 - Техническое Задание

Приложение 2 - описание программы;

Приложение 3 - руководство пользователя;

Приложение 4 - программа и методика испытаний;

Приложение 5 - Графическая часть на 9-12 листах формата А4:

- Структурная схема программы.

- Структура протокольных блоков данных.

- Структурные схемы основных процедур взаимодействия объектов по разработанным протоколам.

- Временные диаграммы работы протоколов.

- Граф диалога пользователя.

- Алгоритмы программ.

8.2. Папка с технической и программной документацией в формате:

<группа>_<Фамилия И.О. студента>_КР_СТ_в_АСОИУ.zip.

9. Порядок приёмки работы:

Приёмка работы осуществляется в соответствии с "Программой и методикой испытаний."

Работа защищается перед комиссией преподавателей кафедры.

10. Дополнительные условия:

Данное Техническое Задание может дополняться и изменяться в установленном порядке.

ОПИСАНИЕ ПРОГРАММЫ

1. Общие сведения.

Наименование: “Программа отправки сообщений через com-порты Чат”.

Программа выполняется на языке программирования С# и работает под управлением операционной системы Windows 98 и выше.

2. Назначение разработки.

Программа должна реализовывать функцию передачи текстовых файлов между двумя ПЭВМ, соединенными через интерфейс RS-232C нуль-модемным кабелем.

4. Используемые технические средства.

Программа должна работать на IBM-совместимой ЭВМ следующей конфигурации:

1. Центральный процессор Pentium I или выше;
2. Объем оперативной памяти 32 Мб;
3. Видеоадаптер и монитор VGA и выше;
4. Стандартная клавиатура;
5. Свободного пространства на жестком диске 2Мб;

Для работы программы требуются два IBM-совместимых компьютера, соединенных нуль-модемным кабелем через интерфейс RS-232C.

5. Входные и выходные данные.

5.1. Входные данные.

Входными данными является текстовое сообщение, набранное пользователем.

5.2. Выходные данные.

Выходными данными являются:

1. текст переданного сообщения на ПЭВМ;
2. сообщения об ошибках и выполнении передачи.

6. Спецификация данных.

6.1. Внутренние данные.

Данные указаны без учета стартовых и стоповых байтов.

Запрос на соединение:

Наименование	Тип поля	Размер (байт)
UPLINK	Byte	1

Поддержание соединения:

Наименование	Тип поля	Размер (байт)
LINKACTIVE	Byte	1

Положительная квитанция:

Наименование	Тип поля	Размер (байт)
ACK	Byte	1

Разрыв соединения:

Наименование	Тип поля	Размер (байт)
DOWNLINK	Byte	1

Информационный блок:

Наименование	Тип поля	Размер (байт)
DAT	Byte	1
Data	AnsiString	Sizeof(Data)

6.2. Структура сообщения.

Программа работает с текстовыми сообщениями стандарта ANSI размером не более 255 символов.

7. Спецификация функций.

`void WriteData(string msg, FrameType CurrentFrameType, bool msg_no_display)`

– процедура записи в порт;

`void DisplayData(MessageType type, string msg)` – процедура вывода данных с порта на экран;

`bool OpenPort()` – функция открытия порта;

`void SetParityValues(object obj)` – процедура заполнения выпадающего списка «Бит четности» в форме «Чат» значениями бита четности;

`void SetStopBitValues(object obj)` - процедура заполнения выпадающего списка «Стопбиты» в форме «Чат» значениями количества стоповых бит;

`void SetPortNameValues(object obj)` - процедура заполнения выпадающего списка «Порт» в форме «Чат» именами доступных COM-портов;

`void comPort_DataReceived(object sender, SerialDataReceivedEventArgs e)` – процедура, которая вызывается, когда в буфере появляются данные;

bool ClosePort() – функция закрытия порта;
 void cmdOpen_Click(object sender, EventArgs e) – процедура обработки нажатия кнопки «Открыть порт» в форме «Чат»;
 void cmdClose_Click(object sender, EventArgs e) - процедура обработки нажатия кнопки «Закрыть порт» в форме «Чат»;
 void cmdConnect_Click(object sender, EventArgs e) – процедура обработки нажатия кнопки «Подключиться» в форме «Чат»;
 void cmdDisconnect_Click(object sender, EventArgs e) – процедура обработки нажатия кнопки «Разъединить» в форме «Чат»;
 void cmdSend_Click(object sender, EventArgs e) – процедура обработки нажатия кнопки «Отправить» в форме «Чат»;
 void frmMain_Load(object sender, EventArgs e) – процедура обработки события загрузки формы «Чат»;
 void SetDefaults() – процедура установки параметров соединения по умолчанию;
 void LoadValues() – процедура загрузки параметров COM-порта;
 void SetControlState() – процедура установки состояний органов управления при первой загрузке формы «Чат»;
 void Tick(object sender, EventArgs e) – процедура обработки события истечения интервала ожидания кадра ACK_UPLINK;
 void Tick_mytimer1(object sender, EventArgs e) – процедура обработки события истечения интервала ожидания кадра ACK_DOWNLINK;
 void Tick_mytimer2(object sender, EventArgs e) - процедура обработки события монитора активности соединения Назначение обработчика - отправка по истечении интервала кадра ACTIVELINK и монитор соединения (монитор статуса comm.LinkActive);
 void rtbDisplay_TextChanged(object sender, EventArgs e) – процедура обработки события изменения текста в диалоговом окне на форме «Чат», синхронна с событием comm.DataReceived(исключая случаев "Port Opened AT DateTime.NOW" и отображения события отправки кадров);
 void frmMain_FormClosing(object sender, FormClosingEventArgs e) – процедура обработки события во время закрытия формы «Чат»;
 void frmMain_FormClosed(object sender, FormClosedEventArgs e) - – процедура обработки события закрытия формы «Чат»;
 void акваToolStripMenuItem_Click(object sender, EventArgs e) – процедура обработки нажатия пункта меню «Аква» меню «Вид»;
 void золотойToolStripMenuItem_Click(object sender, EventArgs e) – процедура обработки нажатия пункта меню «Золотой» меню «Вид»;
 void стандартныйToolStripMenuItem_Click(object sender, EventArgs e) – процедура обработки нажатия пункта меню «Стандартный» меню «Вид»;
 void toolStripMenuItem4_Click(object sender, EventArgs e) – процедура обработки нажатия пункта меню «О программе» меню «Справка»;
 void открытьФайлИсторииToolStripMenuItem_Click(object sender, EventArgs e) - процедура обработки нажатия пункта меню «Открыть файл истории» меню «История»;
 void openFileDialog1_FileOk(object sender, EventArgs e) – процедура открытия окна-диалога для выбора файла;
 void button1_Click(object sender, EventArgs e) – процедура обработки нажатия кнопки «Ок» на форме «О программе»;
 void Form3_Load(object sender, EventArgs e) – процедура обработки события загрузки формы «История»;
 void HammCode(byte a, byte n, out byte code) - процедура кодирования кодовой комбинации кодом Хэмминга.
 int CountBit(byte a) - функция перевода чисел из десятичного представления в двоичное;

`bool HammDecode(byte a,out byte code,byte n)` – процедура декодирования;
`List<byte> CodingStr(string str)` – функция кодирования строки текста;
`string DeCodingStr(byte[] str)` – функция декодирования исходной строки байтов в строку, введенную пользователем.

Листинг основных функций

Connection.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Ports;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WahChat
{
    class Connection
    {
        public bool isPortsOpened = false;

        private byte boundByte = 0xFF;

        private SerialPort incomePort;
        private SerialPort outcomePort;

        public bool isMaster;

        private List<byte> bytesBuffer = new List<byte>();

        public Connection(string incomePortName, string outcomePortName, bool
isMaster)
        {
            this.isPortsOpened = OpenPorts(incomePortName, outcomePortName,
isMaster);

            // ..
        }

        /// <summary>
        /// Открытие портов
        /// </summary>
```

```

private bool OpenPorts(string incomePortName, string outcomePortName, bool
isMaster)
{
    // Создаем объекты портов.
    this.incomePort = new SerialPort(incomePortName);
    this.outcomePort = new SerialPort(outcomePortName);

    this.isMaster = isMaster;

    // Настраиваем порты.
    this.incomePort.Parity = Parity.Even;
    this.incomePort.Handshake = Handshake.RequestToSend;
    this.incomePort.BaudRate = 9600;
    //this.incomePort.ReadBufferSize = 4 * 1024; // TODO: Надо пересчитать
размер буфера.
    this.incomePort.DataReceived += new
SerialDataReceivedEventHandler(RecieveBytes);

    this.outcomePort.Parity = Parity.Even;
    this.outcomePort.Handshake = Handshake.RequestToSend;
    this.outcomePort.BaudRate = 9600;

    // Открываем порты.
    try
    {
        this.incomePort.Open();
        this.outcomePort.Open();
    } catch (Exception ex)
    {
        //Здесь должна быть ошибка невозможности использовать COM-порты
        throw ex;
    }

    return (this.incomePort.IsOpen && this.outcomePort.IsOpen);
}

/// <summary>
/// Закрытие портов
/// </summary>
private bool ClosePorts()
{
    // Закрываем порты.
    this.incomePort.Close();
    this.outcomePort.Close();
}

```



```

        return (this.incomePort.IsOpen && this.outcomePort.IsOpen);
    }

    /// <summary>
    /// Пересылка байтов
    /// </summary>
    public void SendBytes(List<byte> list)
    {
        // TODO: Кодирование
        List<byte> hamm = list;

        // Делаем так, чтобы внутри кадра не встречалось boundByte.
        List<byte> safeList = new List<byte>(hamm.Count);
        foreach (var b in hamm)
        {
            if ((b & 0x7F) == 0x7F)
            {
                safeList.Add(0x7F);
                safeList.Add((byte)(b & 0x80));
            }
            else
            {
                safeList.Add(b);
            }
        }

        // Добавляем стартовый и конечный байт
        safeList.Insert(0, boundByte);
        safeList.Add(boundByte);

        if (this.outcomePort.WriteBufferSize - this.outcomePort.BytesToWrite <
            safeList.Count)
        {
            // Если сообщение не влезло в порт, то надо что-то с этим делать.
            // То ли очередь придумать, то ли ещё что.
            return;
        }

        byte[] arr = safeList.ToArray();
        this.outcomePort.Write(arr, 0, arr.Length);
    }

```

```

    /// <summary>
    /// Получение байтов
    /// </summary>
    public void RecieveBytes(object sender, SerialDataReceivedEventArgs e)
    {
        int bytes = incomePort.BytesToRead;
        byte[] comBuffer = new byte[bytes];

        // Записываем в массив данные от ком порта.
        incomePort.Read(comBuffer, 0, bytes);

        foreach (byte incomeByte in comBuffer)
        {
            if (incomeByte == boundByte)
            {
                if (this.bytesBuffer.Count > 0)
                {
                    NetworkService.GetSharedService().HandleMessage(this.bytesBuffer);
                }

                this.bytesBuffer = new List<byte>();
            }
            else
            {
                this.bytesBuffer.Add(incomeByte);
            }
        }
    }
}

```

HammerCoder.cs (кодирование и декодирование)

```

using System;
using System.Collections.Generic;
using System.Text;

namespace WahChat
{
    class HammerCoder
    {
        static void HammerCode(byte a, byte n, out byte code)

```

```

{
    a <<= 3;

    while (CountBit(a) != CountBit(n)) n <<= 1;
    code = a;
    do
    {
        a ^= n;
        do
        {
            n >>= 1;
        }
        while (CountBit(a) != CountBit(n));
    } while (a > 7);
    code |= a;
}

static int CountBit(byte a)
{
    int count = 0;
    while (!(a == 0))
    {
        a /= 2;
        ++count;
    }

    return (count);
}

static bool HammDecode(byte a, out byte code, byte n)
{
    while (CountBit(a) != CountBit(n)) n <<= 1;
    code = a;
    do
    {
        code ^= n;
        do
        {
            n >>= 1;
        }
        while (CountBit(code) != CountBit(n));
    }
    while (code > 7);
}

```

```

    if (code == 0)
    {
        code = (byte) (a >> 3);
        return (true);
    }
    else return (false);
}

static public List<byte> CodeString(string str)
{
    char[] ChrArr;
    byte LShort, HShort;
    byte LWord, HWord;
    byte Gx = 11;
    byte Code;
    List<byte> CodingString = new List<byte>();

    ChrArr = str.ToCharArray();

    short[] ShortArr = new short[str.Length];

    for (int i = 0; i < ChrArr.Length; i++)
    {
        ShortArr[i] = (short)ChrArr[i];
    }

    for (int i = 0; i < ShortArr.Length; i++)
    {
        LShort = (byte) (ShortArr[i] & 0x00FF);
        HShort = (byte) ((ShortArr[i] & 0xFF00) >> 8);

        LWord = (byte) (LShort & 0x0F);
        HWord = (byte) ((LShort & 0xF0) >> 4);

        HammCode(LWord, Gx, out Code);
        CodingString.Add(Code);
        HammCode(HWord, Gx, out Code);
        CodingString.Add(Code);

        LWord = (byte) (HShort & 0x0F);
        HWord = (byte) ((HShort & 0xF0) >> 4);
    }
}

```

```

        HammCode(LWord, Gx, out Code);
        CodingString.Add(Code);
        HammCode(HWord, Gx, out Code);
        CodingString.Add(Code);
    }

    return (CodingString);
}

/// <summary>
static public string DecodeString(byte[] str)
{
    char ch;
    string DecodedString = "";
    byte LWord, HWord; // младшая и старшие части символа
    short LShort, HShort;

    byte Gx = 11;

    for (int i = 0; i < str.Length; i++)
    {
        if (!HammDecode(str[i], out LWord, Gx)) return ("");
        if (!HammDecode(str[++i], out HWord, Gx)) return ("");

        LShort = (short)((HWord << 4) | LWord);

        if (!HammDecode(str[++i], out LWord, Gx)) return ("");
        if (!HammDecode(str[++i], out HWord, Gx)) return ("");
        HShort = (short)((HWord << 4) | LWord);

        ch = Convert.ToChar((HShort << 8) | LShort);
        Console.WriteLine("ch={0}", ch);
        DecodedString += ch;
    }

    return (DecodedString);
}
}
}

```

Frame.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WahChat
{
    class Frame
    {
        /// <summary>
        /// Тип кадра
        /// </summary>
        public enum Type: byte
        {
            /// Установка логического соединения.
            Link,
            /// Получение списка пользователей.
            Ask,
            /// Отправка сообщений пользователя.
            Data,
            /// Отправка запроса на переотправку сообщения.
            Error,
            /// Разъединение соединения.
            Downlink
        }

        public Type type;
        public List<byte> data;

        public int authorID;
        public string message;

        public Frame()
    }
}
```

```

{
    // ..
}

public Frame(List<byte> data)
{
    this.data = data;

    byte typeByte = data[0];
    switch (typeByte)
    {
        case (byte)Type.Link:
            this.type = Type.Link;
            break;

        case (byte)Type.Ask:
            this.type = Type.Ask;
            break;

        case (byte)Type.Data:
            this.type = Type.Data;
            this.authorID = (int)data[1];

            byte[] byteArray = data.ToArray();

            int messageLength = data.Count - 2;
            byte[] messageData = new byte[messageLength];

            Array.Copy(byteArray, 2, messageData, 0, messageLength);

            this.message = System.Text.Encoding.UTF8.GetString(messageData,
0, messageData.Length);

            break;

        case (byte)Type.Error:
            this.type = Type.Error;
            break;

        case (byte)Type.Downlink:
            this.type = Type.Downlink;
            break;
    }
}

```

```

        default:
            this.type = Type.Link;
            break;
    }

}

public Frame(Type type)
{
    List<byte> data = new List<byte>();
    data.Add((byte)type);
    this.data = data;
}
}
}

```

NetworkService.cs

```

using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Ports;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using WahChat.Messages;

namespace WahChat
{
    class NetworkService
    {
        public Label notificationLabel;
        public Button connectButton;
        public ListBox chatBox; //Отправленные со станции сообщения
        public ListBox chatInBox; //Поступившие на станцию сообщения от других станций

        private NetworkService()
        {
            // ..
        }
    }
}

```



```

        private static readonly NetworkService _sharedService = new
NetworkService();

        public static NetworkService GetSharedService()
        {
            return _sharedService;
        }

        /// Текущее соединение
        public Connection currentConnection;

        /// Текущая сессия
        public Session currentSession;

        /// <summary>
        /// Создание соединения
        /// </summary>
        public void CreateConnection(string incomePortName, string outcomePortName,
bool isMaster)
        {
            this.currentConnection = new Connection(incomePortName, outcomePortName,
isMaster);

            // формирование LINK кадра..
            // отправка LINK кадра..
        }

        /// <summary>
        /// Закрытие соединения
        /// </summary>
        public void CloseConnection()
        {
            Frame frame = new Frame(Frame.Type.Downlink);
            this.SendFrame(frame);
        }

        /// <summary>
        /// Обработка пришедшего сообщения
        /// </summary>
        public void HandleMessage(List<byte> message)
        {
            Frame frame = new Frame(message);
            this.HandleFrame(frame);
        }

```

```

    }

    /// <summary>
    /// Обработка пришедшего кадра
    /// </summary>
    public void HandleFrame(Frame frame)
    {
        switch (frame.type)
        {
            case Frame.Type.Link:

                this.notificationLabel.Invoke((MethodInvoker)delegate {

                    // Running on the UI thread
                    this.notificationLabel.Text = "Соединение установлено";
                    this.connectButton.Text = "Войти";
                });

                // Если станция не ведущая, то отправляем дальше
                if (currentConnection.isMaster == false)
                {
                    this.SendFrame(frame);
                }

                break;

            case Frame.Type.Ask:

                // Если станция не ведущая, то отправляем дальше
                if (currentConnection.isMaster == false)
                {
                    this.SendFrame(frame);
                }

                break;

            case Frame.Type.Data:

                ChatMessage msg = new
                ChatMessage(DateTime.Now.ToString("hh:mm"), frame.authorID, frame.message);

                // Сравниваем ID сессии и автора сообщения. В зависимости от
                него помещаем сообщение во входящие или отправленные.

                if (NetworkService.GetSharedService().currentSession.username ==
                msg.authorID)

```

```

{
    this.chatBox.Invoke((MethodInvoker)delegate {

        // Running on the UI thread
        this.chatBox.Items.Add(msg.ToString());
    });
} else
{
    this.chatInBox.Invoke((MethodInvoker)delegate {

        // Running on the UI thread
        this.chatInBox.Items.Add(msg.ToString());
    });
}

// Если станция не является отправителем, то отправляем дальше
if (currentSession.username != frame.authorID)
{
    this.SendFrame(frame);
}

break;

case Frame.Type.Error:

    // Если станция не ведущая, то отправляем дальше
    if (currentConnection.isMaster == false)
    {
        this.SendFrame(frame);
    }

    break;

case Frame.Type.Downlink:

    // Если станция не ведущая, то отправляем дальше
    if (currentConnection.isMaster == false)
    {
        this.SendFrame(frame);
    }

    System.Windows.Forms.Application.Exit();
}

```

```

        break;
    }
}

public void SendFrame(Frame frame)
{
    this.currentConnection.SendBytes(frame.data);
}

public void SendMessage(string message)
{
    byte[] byteStr = System.Text.Encoding.UTF8.GetBytes(message);

    List<byte> data = new List<byte>();

    data.Add((byte)Frame.Type.Data);
    data.Add((byte)currentSession.username);

    foreach (byte b in byteStr)
    {
        data.Add(b);
    }

    Frame frame = new Frame();
    frame.data = data;

    this.SendFrame(frame);
}

/// <summary>
/// Список доступных портов
/// </summary>
public string[] GetPortsNames()
{
    return SerialPort.GetPortNames();
}

/// <summary>
/// Создание сессии
/// </summary>
public void CreateSession(int username)
{
    this.currentSession = new Session(username);
}

```

```

        // формирование кадра с username..
        // отправка кадра с username..
    }

    /// <summary>
    /// Закрытие сессии
    /// </summary>
    public void CloseSession()
    {
        // ..
    }
}

```

Session.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WahChat
{
    class Session
    {
        public int username;

        public Session(int username)
        {
            this.username = username;
        }
    }
}

```

Login.cs (форма авторизации и подключения)

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WahChat
{
    public partial class Login : Form
    {
        public Login()
        {
            InitializeComponent();

            NetworkService.GetSharedService().notificationLabel =
this.notificationLabel;
            NetworkService.GetSharedService().connectButton = this.loginButton;

            this.loginButton.Text = "Подключиться";

            if (NetworkService.GetSharedService().currentConnection.isPortsOpened)
            {
                this.notificationLabel.Text = "Порты открыты";
            }
            else
            {
                this.notificationLabel.Text = "Ошибка открытия портов";
            }
        }

        private void loginButton_Click(object sender, EventArgs e)
        {
            int username = int.Parse(textBox.Text);
            NetworkService.GetSharedService().CreateSession(username);

            Frame frame = new Frame(Frame.Type.Link);
            NetworkService.GetSharedService().SendFrame(frame);

            this.notificationLabel.Text = "Отправка..";
        }
    }
}

```

```

        this.Hide();

        Chat chatForm = new Chat();
        chatForm.Show();
    }

    private void Login_Load(object sender, EventArgs e)
    {

    }
}

```

Setup.cs (форма настроек соединения)

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WahChat
{
    public partial class Setup : Form
    {
        public Setup()
        {
            InitializeComponent();
        }
    }
}

```

```

private void Setup_Load(object sender, EventArgs e)
{
    // Показываем список COM-портов.
    string[] portNames = NetworkService.GetSharedService().GetPortsNames();
    foreach (string portName in portNames)
    {
        incomePortBox.Items.Add(portName);
        outcomePortBox.Items.Add(portName);
    }

    //Добавим список скоростей COM-портов
    box_speed1.Items.Clear();
    box_speed2.Items.Clear();
    box_speed1.Items.Add(21600);
    box_speed2.Items.Add(21600);
}

private void connectButton_Click(object sender, EventArgs e)
{
    // Проверяем, выставлены ли порты.
    if ((incomePortBox.SelectedItem != null) && (outcomePortBox.SelectedItem
!= null))
    {
        string incomePort = incomePortBox.SelectedItem.ToString();
        string outcomePort = outcomePortBox.SelectedItem.ToString();

        // Если порты одинаковые, то показываем ошибку.
        if (incomePort == outcomePort)
        {
            MessageBox.Show("Выберите различные COM-порты", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            return;
        }

        // Стартуем соединение.
        NetworkService.GetSharedService().CreateConnection(incomePort,
outcomePort, checkBox.Checked);

        this.Hide();

        Login loginForm = new Login();
        loginForm.Show();
    }
}

```



```

        else
        {
            MessageBox.Show("Оба COM-порта должны быть выбраны", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
    }
}

```

Chat.cs (форма чата)

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WahChat
{
    public partial class Chat : Form
    {
        public Chat()
        {
            InitializeComponent();

            NetworkService.GetSharedService().chatBox = this.chatBox; //Отправленные
            NetworkService.GetSharedService().chatInBox = this.chatInbox; //Входящие

            this.usernameLabel.Text = String.Format("Подключен как \"{0}\"",
            NetworkService.GetSharedService().currentSession.username);
            this.chatBox.SelectionMode = SelectionMode.None;
            this.chatInbox.SelectionMode = SelectionMode.None;
        }

        private void SendButton_Click(object sender, EventArgs e)
        {
            string message = messageBox.Text;
            NetworkService.GetSharedService().SendMessage(message);
        }
    }
}

```

```

        messageBox.Clear();
    }

    private void closeButton_Click(object sender, EventArgs e)
    {
        NetworkService.GetSharedService().CloseConnection();
    }

    private void Chat_Load(object sender, EventArgs e)
    {

    }

    private void button_about_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Выполнена в рамках курса \"Сетевые технологии в АСОИУ\" \nИсполнители:\tКрротенко Е.А. ИУ5Ц-81Б\n\t\tПылаев Б.А. ИУ5Ц-82Б\n\t\tЧиварзин А.Е. ИУ5Ц-82Б\nПреподаватель:\tГалкин В.А.", "Разработка");
    }
}

```

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

1. Назначение программы

Программа «Чат» предназначена для обмена текстовыми сообщениями между двумя компьютерами, соединенными через интерфейс RS-232C.

2. Условия выполнения программы

Для нормальной работы программы требуется IBM-совместимый компьютер с микропроцессором Pentium I или выше, 32MB оперативной памяти, соединенные нуль-модемным кабелем через интерфейс RS-232C. Также требуется оператор (пользователь), имеющий базовые знания о работе с компьютером и операционной системой Windows98 и выше.

3. Выполнение программы

3.1. Инсталляция/деинсталляция

Инсталляция программы заключается в копировании файла ChatClient.exe. \nДля деинсталляции нужно удалить этот файл.

3.2. Запуск программы

На обоих компьютерах из каталога программы, из группы программы в меню «Пуск» или через ярлык на рабочем столе, если он был создан при установке, запустите файл ChatClient.exe.

При этом на экране появится окно выбора портов компьютера первого пользователя (Рисунок 1):

Настройка соединения

	Порт	Скорость передачи
Входящий COM-порт	COM5	21600
Исходящий COM-порт	COM6	21600

Начать соединение

☐ Ведущая станция

Рисунок 1.

После выбора портов и нажатия кнопки «Начать соединение» открывается главное окно программы.

3.3. Главное окно программы.

После ввода портов пользователь выбирает своё имя для входа в чат и нажимает кнопку «Подключиться», которая спровоцирует установку соединения с устройством другого пользователя (Рисунок 2):

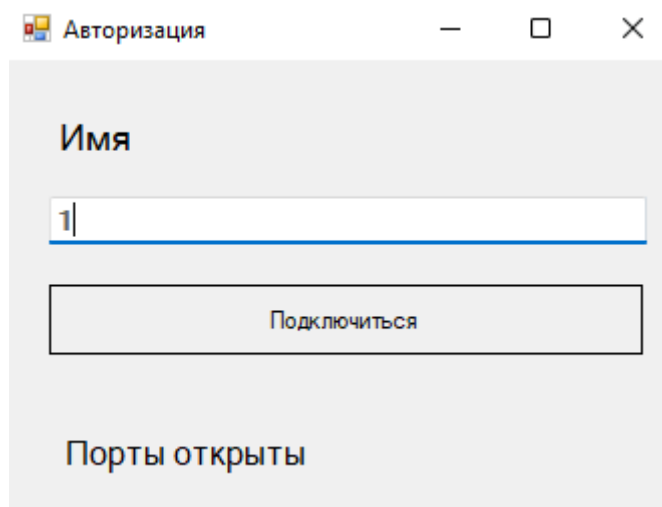


Рисунок 2.

3.4. Главное диалоговое окно программы.

После запуска программы на двух компьютерах, открытия портов, ввода имён пользователями и подключения, между устройствами устанавливается соединение.

После того как соединение установлено, пользователи могут обмениваться сообщениями посредством главного диалогового окна (активизируется кнопка «Отправить») (Рисунки 3а и 3б):

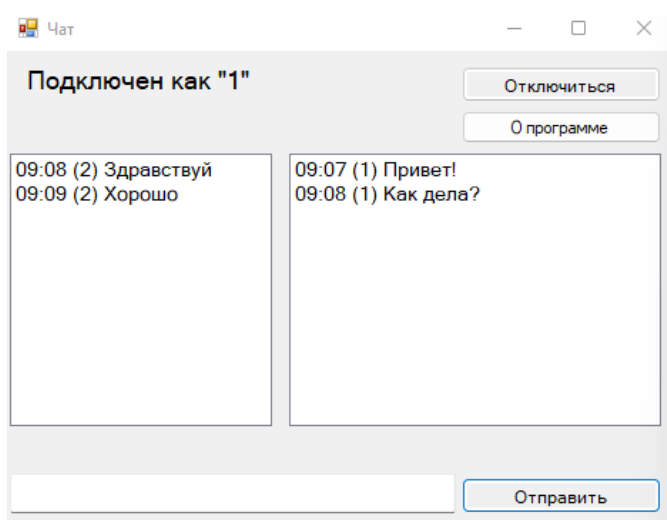


Рисунок 3а.

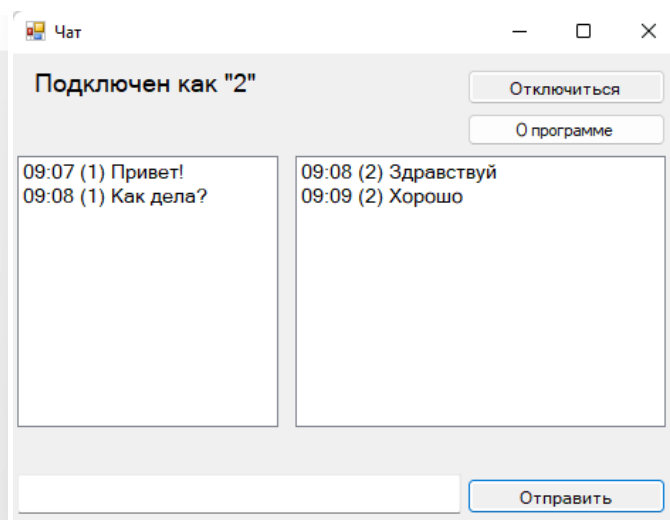


Рисунок 3б.

Для отправки сообщения надо нажать кнопку «Отправить». После чего сообщение будет передано другому пользователю, а также оно будет отображено в секции программы «Чат», представляющей из себя историю диалога (рисунок 4а). При этом в окне с диалогом у принимающей стороны появится сообщение (рисунок 4б):

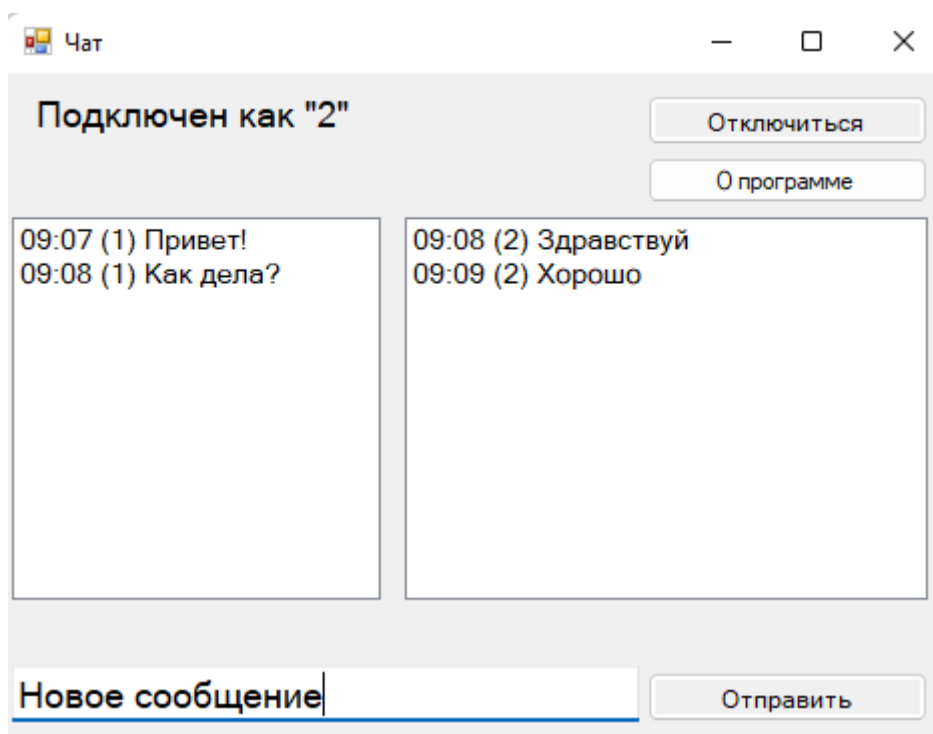


Рисунок 4а.

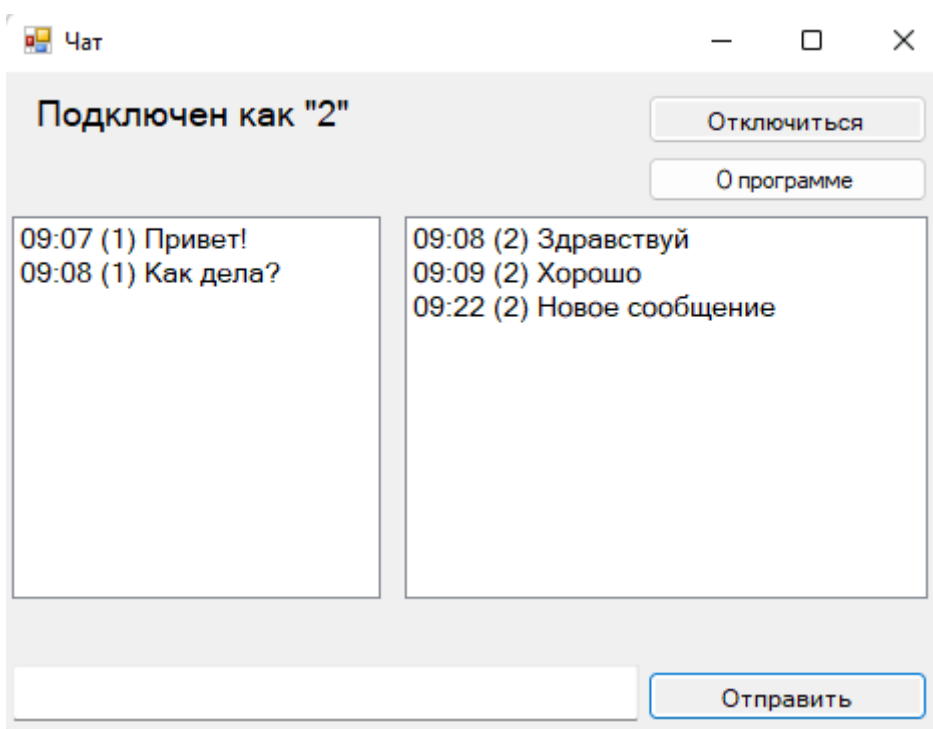


Рисунок 4б.

Для просмотра информации о программе необходимо нажать на кнопку “О программе”. На экране будет выведено информационное сообщение, содержащее наименование курса, ФИО разработчиков (исполнители), ФИО преподавателя (Рисунок 5).

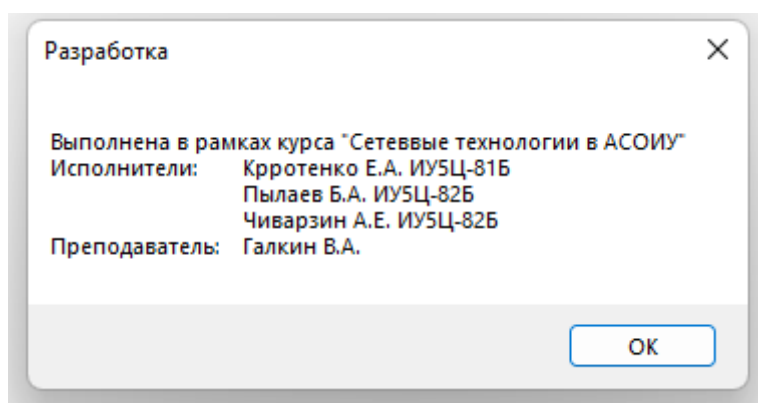


Рисунок 5.

После того как пользователи решат прекратить общение и разорвать соединение, они должны нажать кнопку «Отключиться», что спровоцирует разрыв соединения между устройствами и закрытия COM-портов (Рисунки 6а и 6б).

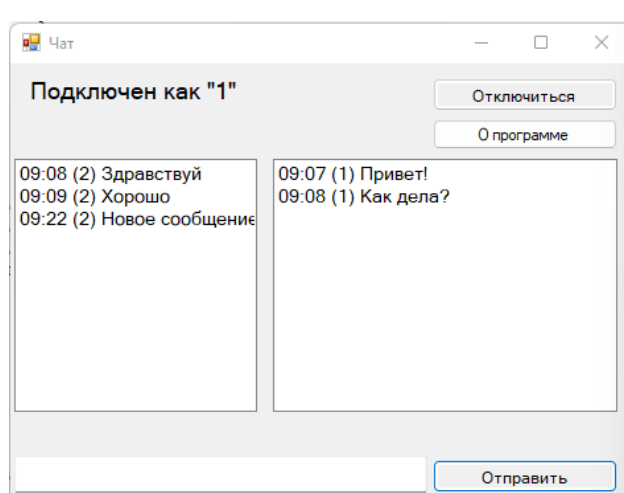


Рисунок 6а.

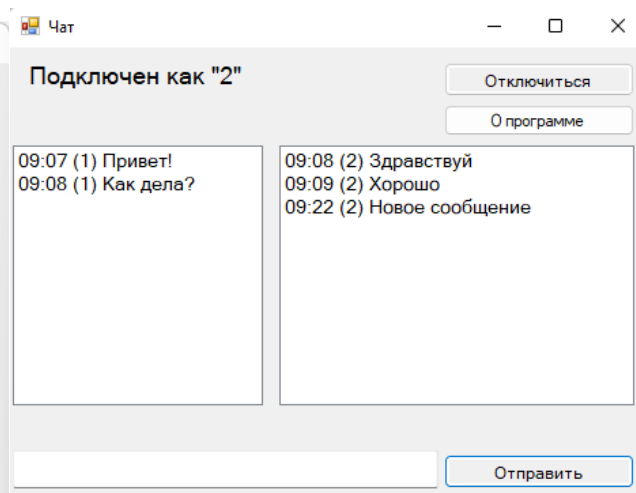


Рисунок 6б.

4. Устранение неполадок

Сообщения об ошибках

Поскольку Windows – не однозадачная система, возможны случаи, когда некоторые программы забирают все ресурсы компьютера, останавливая выполнение всех других программ. В этом случае удаленная машина может «потерять» вашу программу, так как она не отвечает на вызов по истечению таймута. Это может случиться также и при очень плохом качестве связи между компьютерами, например при большой длине кабеля и высоком уровне помех, и при перегрузке буферов com-портов. Во всех перечисленных случаях исходящие сообщения не будут доставляться до конечного пользователя.

В случае возникновения непредвиденных ошибок запускается стандартный обработчик ошибок Windows.

ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

1. Объект испытаний.

Объектом испытания является коммуникационная программа, предназначенная для приёма/передачи сообщений между компьютерами, соединёнными нуль-модемным кабелем через интерфейс RS-232C.

2. Цель испытаний.

Целью проведения испытаний является доказательство работоспособности описанного в пункте 1 объекта испытаний.

3. Требования к объекту испытаний.

Требования к объекту испытаний представлены в документе «Техническое задание».

4. Требования к программной документации.

Во время проведения испытания должны быть представлены следующие три документа:

- а) Техническое задание
- б) Программа и методика испытаний
- с) Описание программы

5. Средства и порядок испытаний.

Для проведения испытания необходимы два компьютера, удовлетворяющие требованиям, описанным в п.6 документа «Техническое задание», соединённых нуль-модемным кабелем через интерфейс RS-232C через порты COM. Также на каждом компьютере должны располагаться файлы представляемой программы.

Если для испытания используются виртуальные машины, соединенные с помощью виртуально реализованного COM-порта, то на компьютере уже должны быть установлены виртуальные машины. И следует выполнить следующие действия:

1. Запустить программу Virtual Null Modem и в ней создать нульмодемное соединение «с полным контролем передачи 2» для двух новых виртуальных портов (например COM5, COM6).
2. Запустить консоль для запуска виртуальных машин. В настройках двух виртуальных машин настроить порт COM1 на виртуальный порт COM5 и COM6 для каждой виртуальной машины соответственно.
3. Запустить обе виртуальные машины.

Дальнейшая методика испытаний проводится на уже запущенной виртуальной машине.

Программа испытаний для проверки работоспособности испытуемой программы:

№	№ пункта ТЗ	Проверяемая функция	Выполняемые действия	Результат
1		Запуск программы	Запустить файл ChatClient.exe	Открыто окно выбора портов компьютера первого пользователя
2	5.2.2	Выбор портов и скорости передачи	Выбрать порты и скорость передачи и нажать кнопку «Начать соединение»	Открыто главное окно программы
3	5.2.3	Соединение с устройством другого пользователя	Ввести имя пользователя и нажать кнопку «Подключиться»	Устанавливается соединение
4	5.2.4	Отправить сообщение	После установки соединения активизируется кнопка «Отправить» на главном окне. Для отправки сообщения надо ввести его и нажать кнопку «Отправить».	После этого сообщение будет передано другому пользователю, а также оно будет отображено в секции программы «Чат». При этом в окне с диалогом у принимающей стороны появится сообщение
5	5.2.3	Прекращение общения	Необходимо нажать кнопку «Отключиться», что спровоцирует разрыв соединения между устройствами и закрытия COM-портов	После нажатия кнопки «Отключиться» соединение будет разорвано и COM порты будут закрыты.

ГРАФИЧЕСКАЯ ЧАСТЬ

Графическая часть в составе Структурная схема программы, Структура протокольных блоков данных, Структурные схемы основных процедур взаимодействия объектов по разработанным протоколам, Временные диаграммы работы протоколов, Граф диалога пользователя, Алгоритмы программ представлена на отдельных листах А4.