Московский государственный технический университет им. Н.Э. Баумана

Утверждаю:
Галкин В.А. "__"_____2020 г.

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

Пояснительная записка
(вид документа)

писчая бумага
(вид носителя)

20
(количество листов)

ИСПОЛНИТЕЛИ:

студенты группы ИУ5-61Б
Матиенко А.П. _____
Молева А.А. _____
Белоусов Е.А. _____

Москва - 2020 г.

Содержание

1. Введение

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

2. Требования к программе

К программе предъявляются следующие требования. Программа должна:

2.1. Устанавливать соединение между компьютерами и контролировать его целостность

2.2. Обеспечивать правильность передачи и приема данных с помощью кодирования пакета по коду Хемминга

2.3. Обеспечивать функцию передачи сообщений

2.4. Обеспечивать функцию широковещательной отправки сообщений

Программа выполняется под управлением OS Windows XP и выше, а также OS Linux. Было решено выполнить реализацию программы с помощью среды разработки Python с применением функционала языка C.

3. 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.

4. Физический уровень.

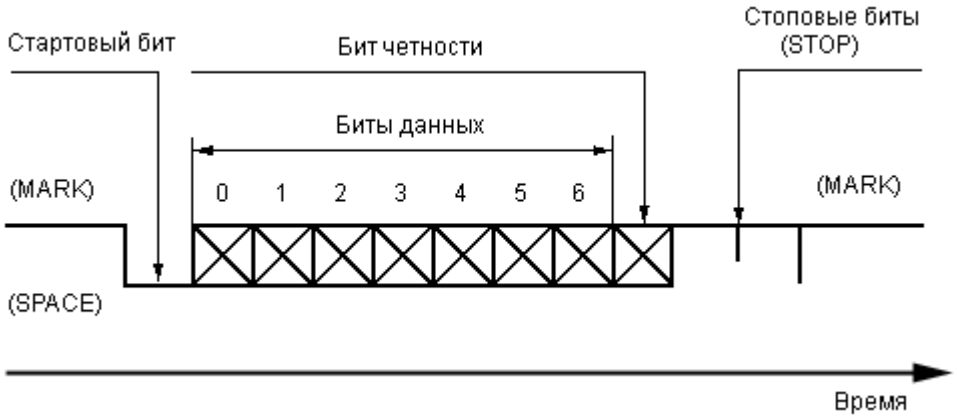
4.1 Функции физического уровня.

Основными функциями физического уровня являются:

1. Задание параметров COM-порта.
2. Установление физического канала.
3. Поддержание соединения.
4. Разъединение физического канала.
5. Передача информации из буфера в интерфейс.
6. Прием информации и ее накопление в буфере.

4.2 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* (см. рисунок). 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
3	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.

4.3 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 справен, 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**.

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-1;
6. TD-1 — RD-1;
7. SG-1 — SG-2;

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

4.4 Настройка COM-порта средствами Python и C.

Язык программирования Python содержит модуль PySerial. Этот модуль инкапсулирует доступ для последовательного порта. Он предоставляет бэкенды для Python, работающего в Windows и Linux. PySerial использует прикладной программный интерфейс Win32 API, который предлагает широкие возможности по настройке COM-порта.

4.4.1 PySerial API

1. **port** – имя устройства or **None**.
2. **baudrate** (*int*) – частота бод(пропускная способность): 9600 или 115200…
3. **bytesize** – количество бит данных. Возможные значения: **FIVEBITS**, **SIXBITS**, **SEVENBITS**, **EIGHTBITS**
4. **parity**–бит четности. Возможные значения: **PARITY_NONE**, **PARITY_EVEN**, **PARITY_ODD** **PARITY_MARK**, **PARITY_SPACE**
5. **stopbits** – номер стоп-бит. Возможные значения: **STOPBITS_ONE**, **STOPBITS_ONE_POINT_FIVE**, **STOPBITS_TWO**
6. **timeout** (*float*) – Set a read timeout value. (Установите значение времени ожидания чтения)
7. **xonxoff** (*bool*) – Enable software flow control. (Включить программное управление потоком)
8. **rtscts** (*bool*) – Enable hardware (RTS/CTS) flow control. (Включить аппаратное (RTS/CTS) управление потоком)
9. **dsrdtr** (*bool*) – Enable hardware (DSR/DTR) flow control. (Включить аппаратное (DSR / DTR) управление потоком)
10. **write_timeout** (*float*) – Set a write timeout value. (Установите значение времени ожидания записи)
11. **inter_byte_timeout** (*float*) – Inter-character timeout, **None** to disable (default).(Межсимвольный тайм-аут, отключить не нужно (по умолчанию).)
12. **exclusive** (*bool*) – Set exclusive access mode (POSIX only). A port cannot be opened in exclusive access mode if it is already open in exclusive access mode.

(Установить эксклюзивный режим доступа (только POSIX). Порт нельзя открыть в режиме исключительного доступа, если он уже открыт в режиме исключительного доступа.)

4.4.2 Структура DCB и функции для работы с ней.

Функция SetCommDCB позволяет настроить COM-порт на основе данных, содержащихся в структуре DCB. Прототип этой функции выглядит так:

```
BOOL SetCommDCB(HANDLE hCom, LPDCB lpDCB);
```

Как видно, функция очень проста и имеет всего два параметра:

hCom — описатель последовательного порта.

lpDCB — указатель на структуру DCB.

Функция GetCommDCB позволяет заполнить структуру DCB данными, соответствующими текущим параметрам COM-порта. Прототип этой функции выглядит так:

```
BOOL GetCommDCB(HANDLE hCom, LPDCB lpDCB);
```

Как видно, функция очень проста и имеет всего два параметра:

hCom — описатель последовательного порта.

lpDCB — указатель на структуру DCB.

Структура содержит основные параметры порта.

```
typedef struct _DCB {
    DWORD DCBlength;
    DWORD BaudRate;
    DWORD fBinary:1;
    DWORD fParity:1;
    DWORD fOutxCtsFlow:1;
    DWORD fOutxDsrFlow:1;
    DWORD fDtrControl:2;
    DWORD fDsrSensitivity:1;
    DWORD fTXContinueOnXoff:1;
    DWORD fOutX:1;
    DWORD fInX:1;
    DWORD fErrorChar:1;
    DWORD fNull:1;
    DWORD fRtsControl:2;
    DWORD fAbortOnError:1;
    DWORD fDummy2:17;
    WORD  wReserved;
    WORD  XonLim;
    WORD  XoffLim;
    BYTE  ByteSize;
    BYTE  Parity;
    BYTE  StopBits;
    char  XonChar;
    char  XoffChar;
    char  ErrorChar;
    char  EofChar;
    char  EvtChar;
    WORD  wReserved1;
} DCB;
```

Кратко рассмотрим значения ее полей:

DCBlength — Задает длину, в байтах, структуры DCB. Используется для контроля корректности структуры при передаче ее адреса в функции настройки порта

BaudRate — Скорость передачи данных. Возможно указание следующих констант: CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800, CBR_9600, CBR_14400, CBR_19200, CBR_38400, CBR_56000, CBR_57600, CBR_115200, CBR_128000, CBR_256000. Как видно, эти константы соответствуют всем стандартным скоростям обмена. На самом деле, это поле содержит числовое значение скорости передачи, а константы просто являются символическими именами.

fBinary — Включает двоичный режим обмена. Win32 не поддерживает недвоичный режим, поэтому данное поле всегда должно быть равно 1, или логической константе TRUE (что предпочтительней).

fParity — Включает режим контроля четности. Если это поле равно TRUE, то выполняется проверка четности, при ошибке, в вызывающую программу, выдается соответствующий код завершения.

fOutxCtsFlow — Включает режим слежения за сигналом CTS. Если это поле равно TRUE и сигнал CTS сброшен, передача данных приостанавливается до установки сигнала CTS. Это позволяет подключенному к компьютеру прибору приостановить поток передаваемой в него информации, если он не успевает ее обрабатывать.

fOutxDsrFlow — Включает режим слежения за сигналом DSR. Если это поле равно TRUE и сигнал DSR сброшен, передача данных прекращается до установки сигнала DSR

fDtrControl — Задает режим управления обменом для сигнала DTR. Это поле может принимать следующие значения:

DTR_CONTROL_DISABLE	Запрещает использование линии DTR
DTR_CONTROL_ENABLE	Разрешает использование линии DTR
DTR_CONTROL_HANDSHAKE	Разрешает использование *рукопожатия* для выхода из ошибочных ситуаций. Этот режим используется, в частности, модемами при восстановленни в ситуации потери связи

fDsrSensitivity — Задает чувствительсть коммуникационного драйвера к состоянию линии DSR. Если это поле равно TRUE, то все принимаемые данные игнорируются драйвером (коммуникационный драйвер расположен в операционной системе), за исключением тех, которые принимаются при установленом сигнале DSR.

fTXContinueOnXoff — Задает, прекращается ли передача при переполнении приемного буфера и передаче драйвером символа XoffChar. Если это поле равно TRUE, то передача продолжается, несмотря на то, что приемный буфер содержит более XoffLim символов и близок к переполнению, а драйвер передал символ XoffChar для приостановления потрока принимаемых данных. Если поле равно FALSE, то передача не будет продолжена до тех пор, пока в приемном буфере не останется меньше XonLim символов и драйвер не передаст символ XonChar для возобновления потока принимаемых данных. Таким образом это поле вводит некую зависимость между управлением входным и выходным потоками информации.

fOutX — Задает использование XON/XOFF управления потоком при передаче. Если это поле равно TRUE, то передача останавливается при приеме символа XoffChar, и возобновляется при приеме символа XonChar.

fInX — Задает использование XON/XOFF управления потоком при приеме. Если это поле равно TRUE, то драйвер передает символ XoffChar, когда в приемном буфере находится более XoffLim, и XonChar, когда в приемном буфере остается менее XonLim символов.

fErrorChar — Указывает на необходимость замены символов с ошибкой четности на символ задаваемый полем ErrorChar. Если это поле равно TRUE, и поле fParity равно TRUE, то выполняется замена.

fNull — Определяет действие выполняемое при приеме нулевого байта. Если это поле TRUE, то нулевые байты отбрасываются при передаче.

fRtsControl — задает режим управления потоком для сигнала RTS. Если это поле равно 0, то по умолчанию подразумевается RTS_CONTROL_HANDSHAKE. Поле может принимать одно из следующих значений:

RTS_CONTROL_DISABLE	Запрещает использование линии RTS
RTS_CONTROL_ENABLE	Разрешает использование линии RTS
RTS_CONTROL_HANDSHAKE	Разрешает использование RTS *рукопожатия*. Драйвер устанавливает сигнал RTS когда приемный буфер заполнен менее, чем на половину, и сбрасывает, когда буфер заполняется более чем на три четверти.
RTS_CONTROL_TOGGLE	Задает, что сигнал RTS установлен, когда есть данные для передачи. Когда все символы из передающего буфера переданы, сигнал сбрасывается.

fAbortOnError — Задает игнорирование всех операций чтения/записи при возникновении ошибки. Если это поле равно TRUE, драйвер прекращает все операции чтения/записи для порта при возникновении ошибки. Продолжать работать с портом можно будет только после устранения причины ошибки и вызова функции ClearCommError.

fDummy2 — Зарезервировано и не используется.

wReserved — Не используется, должно быть установлено в 0.

XonLim — Задает минимальное число символов в приемном буфере перед посылкой символа XON.

XoffLim — Определяет максимальное количество байт в приемном буфере перед посылкой символа XOFF. Максимально допустимое количество байт в буфере вычисляется вычитанием данного значения из размера применого буфера в байтах.

ByteSize — Определяет число информационных бит в передаваемых и принимаемых байтах.

Parity — Определяет выбор схемы контроля четности. Данное поле должно содержать одно из следующих значений:

EVENPARITY	Дополнение до четности
MARKPARITY	Бит четности всегда 1
NOPARITY	Бит четности отсутствует
ODDPARITY	Дополнение до нечетности
SPACEPARITY	Бит четности всегда 0

StopBits — Задает количество стоповых бит. Поле может принимать следующие значения:

ONESTOPBIT	Один стоповый бит
ONE5STOPBIT	Полтора стоповых бита
TWOSTOPBIT	Два стоповых бита

XonChar — Задает символ XON используемый как для приема, так и для передачи.
XoffChar — Задает символ XOFF используемый как для приема, так и для передачи.
ErrorChar — Задает символ, использующийся для замены символов с ошибочной четностью.
EofChar — Задает символ, использующийся для сигнализации о конце данных.
EvtChar — Задает символ, использующийся для сигнализации о событии.
wReserved1 — Зарезервировано и не используется.

4.4.3 Структура COMMTIMEOUTS и функции для работы с ней.

Функция SetCommTimeouts позволяет настроить COM-порт на основе данных, содержащихся в структуре COMMTIMEOUTS. Прототип этой функции выглядит так:

```
BOOL SetCommDCB(HANDLE hCom, LPCOMMTIMEOUTS lpCOMMTIMEOUTS);
```

Как видно, функция очень проста и имеет всего два параметра:
hCom — описатель последовательного порта.
lpCOMMTIMEOUTS — указатель на структуру COMMTIMEOUTS.

Функция GetCommTimeouts позволяет заполнить структуру COMMTIMEOUTS данными, соответствующими текущим параметрам COM-порта. Прототип этой функции выглядит так:

```
BOOL GetCommTimeouts (HANDLE hCom, LPCOMMTIMEOUTS lpCOMMTIMEOUTS);
```

Как видно, функция очень проста и имеет всего два параметра:

hCom — описатель последовательного порта.
lpCOMMTIMEOUTS — указатель на структуру COMMTIMEOUTS.

Структура содержит временные параметры, используемые при операциях ввода/вывода.

```
typedef struct _COMMTIMEOUTS {
      DWORD ReadIntervalTimeout;
      DWORD ReadTotalTimeoutMultiplier;
      DWORD ReadTotalTimeoutConstant;
      DWORD WriteTotalTimeoutMultiplier;
      DWORD WriteTotalTimeoutConstant;
  } COMMTIMEOUTS;
```

Кратко рассмотрим значения ее полей:

ReadIntervalTimeout — Максимальное время, в миллисекундах, допустимое между двумя последовательными символами считываемыми с коммуникационной линии. Во время операции чтения временной период начинает отсчитываться с момента приема первого символа. Если интервал между двумя последовательными символами превысит заданое значение, операция чтения завершается и все данные, накопленые в буфере, передаются в программу. Нулевое значение данного поля означает, что данный тайм-аут не используется. Значение MAXDWORD, вместе с нулевыми значениями полей ReadTotalTimeoutConstant и ReadTotalTimeoutMultiplier, означает

немедленный возврат из операции чтения с передачей уже принятого символа, даже если ни одного символа не было получено из линии.

ReadTotalTimeoutMultiplier — Задает множитель, в миллисекундах, используемый для вычисления общего тайм-аута операции чтения. Для каждой операции чтения данное значение умножается на количество запрошенных для чтения символов.

ReadTotalTimeoutConstant — Задает константу, в миллисекундах, используемую для вычисления общего тайм-аута операции чтения. Для каждой операции чтения данное значение прибавляется к результату умножения ReadTotalTimeoutMultiplier на количество запрошеных для чтения символов. Нулевое значение полей ReadTotalTimeoutMultiplier и ReadTotalTimeoutConstant означает, что общий тайм-аут для операции чтения не используется.

WriteTotalTimeoutMultiplier — Задает множитель, в миллисекундах, используемый для вычисления общего тайм-аута операции записи. Для каждой операции записи данное значение умножается на количество записываемых символов.

WriteTotalTimeoutConstant — Задает константу, в миллисекундах, используемую для вычисления общего тайм-аута операции записи. Для каждой операции записи данное значение прибавляется к результату умножения WriteTotalTimeoutMultiplier на количество записываемых символов. Нулевое значение полей WriteTotalTimeoutMultiplier и WriteTotalTimeoutConstant означает, что общий тайм-аут для операции записи не используется.

4.4.4 Структура COMSTAT и функции для работы с ней.

Функция ClearCommError не только сбрасывает признак ошибки для соответствующего порта, но и возвращает более подробную информацию об ошибке. Кроме того, возможно получение информации о текущем состоянии порта. Прототип этой функции выглядит так:

```
BOOL ClearCommError(
      HANDLE    hFile,
      LPDWORD   lpErrors,
      LPCOMSTAT lpStat
  );
```

Вот что означают параметры:

hFile — Описатель открытого файла коммуникационного порта.

lpErrors — Адрес переменной, в которую заносится информация об ошибке. В этой переменной могут быть установлен один или несколько из следующих бит:

CE_BREAK	Обнаружено состояние разрыва связи
CE_DNS	Только для Windows95. Параллельное устройство не выбрано.
CE_FRAME	Ошибка обрамления.
CE_IOE	Ошибка ввода-вывода при работе с портом
CE_MODE	Запрошенный режим не поддерживается, или неверный описатель hFile. Если данный бит установлен, то значение остальных бит не имеет значение.
CE_OOP	Только для Windows95. Для параллельного порта установлен сигнал "нет бумаги".
CE_OVERRUN	Ошибка перебега (переполнение аппаратного буфера), следующий символ потерян.
CE_PTO	Только для Windows95. Тайм-аут на параллельном порту.

CE_RXOVER	Переполнение приемного буфера или принят символ после символа конца файла (EOF)
CE_RXPARITY	Ошибка четности
CE_TXFULL	Переполнение буфера передачи

lpStat — Адрес структуры COMMSTAT. Должен быть указан, или адрес выделенного блока памяти, или NULL, если не требуется получать информацию о состоянии.

Структура содержит данные о текущем состоянии порта.

```
typedef struct _COMSTAT {
      DWORD fCtsHold:1;
      DWORD fDsrHold:1;
      DWORD fRlsdHold:1;
      DWORD fXoffHold:1;
      DWORD fXoffSent:1;
      DWORD fEof:1;
      DWORD fTxim:1;
      DWORD fReserved:25;
      DWORD cbInQue;
      DWORD cbOutQue;
  } COMSTAT.
```

Кратко рассмотрим значения ее полей:

fCtsHold — Передача приостановлена из-за сброса сигнала CSR.
fDsrHold — Передача приостановлена из-за сброса сигнала DSR.
fRlsdHold — Передача приостановлена из-за ожидания сигнала RLSD (receive-line-signal-detect). Более известное название данного сигнала - DCD (обнаружение несущей).
fXoffHold — Передача приостановлена из-за приема символа XOFF.
fXoffSent — Передача приостановлена из-за передачи символа XOFF. Следующий передаваемый символ обязательно должен быть XON, поэтому передача собственно данных тоже приостанавливается.
fEof — Принят символ конца файла (EOF).
fTxim — В очередь, с помощью TransmitCommChar, поставлен символ для экстреной передачи.
fReserved — Зарезервировано и не используется.
cbInQue — Число символов в приемном буфере. Эти символы приняты из линии но еще не считаны функцией ReadFile.
cbOutQue — Число символов в передающем буфере. Эти символы ожидают передачи в линию. Для синхронных операций всегда 0.

4.4.5 Структура OVERLAPPED.

Структура необходима при выполнении асинхронных операций ввода/вывода и ожидании событий от порта (п. 4.6).

Структура имеет следующее объявление:

```
typedef struct _OVERLAPPED {
      DWORD  Internal;
      DWORD  InternalHigh;
      DWORD  Offset;
      DWORD  OffsetHigh;
      HANDLE hEvent;
  } OVERLAPPED.
```

Кратко рассмотрим значения ее полей:

Internal — Зарезервировано для использования операционной системой. Это поле, которое определяет системозависимый статус, используется, когда функция GetOverlappedResult завершается без установления ошибки в значение ERROR_IO_PENDING.

InternalHigh — Зарезервировано для использования операционной системой. Это поле, которое определяет количество переданных байт, используется, когда функция GetOverlappedResult возвращает TRUE.

Offset — Определяет позицию в файле, с которой начнется передача. Позиция в файле это смещение в байтах от начала файла. Вызывающий процесс устанавливает это поле перед вызовом функций ReadFile и WriteFile. Это поле игнорируется при чтении из или записи в именованный канал или коммуникационное устройство.

OffsetHigh — Определяет старшее слово позиции в файле, с которой начнется передача. Это поле игнорируется при чтении из или записи в именованный канал или коммуникационное устройство.

hEvent — Определяет событие для сигнализации состояния когда передача закончена. Вызывающий процесс устанавливает это поле перед вызовом функций ReadFile, WriteFile, ConnectNamedPipe или TransactNamedPipe.

4.5 Описание функций физического уровня.

4.5.1. Задание параметров COM-порта

В Python для задания параметров используется вызов класса Serial(), в круглые скобки мы записываем значения порта. Используются следующие функции:

```
port(self, port) - название порта(COM3,COM4)
baudrate(self, baudrate) - скорость передачи в бодах
bytesize(self, bytesize) - размер байта
parity(self, parity) - размер четности
stopbits(self, stopbits) - стопбит
```

В Win32 нет функций для непосредственной работы с последовательными и параллельными портами, поэтому с ними работают работают как с файлами. Для этого необходимо воспользоваться функцией открытия файла CreateFile. Ее прототип выглядит так:

```
HANDLE CreateFile(
   LPCTSTR               lpFileName,
   DWORD                 dwDesiredAccess,
   DWORD                 dwShareMode,
   LPSECURITY_ATTRIBUTES lpSecurityAttributes,
   DWORD                 dwCreationDistribution,
   DWORD                 dwFlagsAndAttributes,
   HANDLE                hTemplateFile
);
```

Приведем краткое описание параметров:

lpFileName — Указатель на строку с именем открываемого или создаваемого файла. Последовательные порты имеют имена "COM1", "COM2", "COM3", "COM4" и так далее. Точно так же они назывались в MS-DOS, так что ничего нового тут нет.

dwDesiredAccess — Задает тип доступа к файлу. Возможно использование следующих значений:

0	Опрос атрибутов устройства без получения доступа к нему.
GENERIC_READ	Файл будет считываться.
GENERIC_WRITE	Файл будет записываться.

GENERIC_READ|GENERIC_WRITE	Файл будет и считываться и записываться.

dwShareMode — Задает параметры совместного доступа к файлу. Коммуникационные порты нельзя делать разделяемыми, поэтому данный параметр должен быть равен 0.

lpSecurityAttributes — Задает атрибуты защиты файла. Поддерживается только в Windows NT. Однако при работе с портами должен в любом случае равняться NULL.

dwCreationDistribution — Управляет режимами автосоздания, автоусечения файла и им подобными. Для коммуникационных портов всегда должно задаваться OPEN_EXISTING.

dwFlagsAndAttributes — Задает атрибуты создаваемого файла. Так же управляет различными режимами обработки. Для наших целей этот параметр должен быть или равным 0, или FILE_FLAG_OVERLAPPED. Нулевое значение используется при синхронной работе с портом, а FILE_FLAG_OVERLAPPED при асинхронной, или другими словами, при фоновой обработке ввода/вывода.

hTemplateFile — Задает описатель файла-шаблона. При работе с портами всегда должен быть равен NULL.

4.5.2. Установление/разъединение физического канала.

Открытие/закрытие канала происходит при вызове функции `open()/close()`. В WIN32 используется функция CreateFile

```
HANDLE CreateFile(
   LPCTSTR               lpFileName,
   DWORD                 dwDesiredAccess,
   DWORD                 dwShareMode,
   LPSECURITY_ATTRIBUTES lpSecurityAttributes,
   DWORD                 dwCreationDistribution,
   DWORD                 dwFlagsAndAttributes,
   HANDLE                hTemplateFile
);
```

которая описана выше.

При успешном открытии порта, функция возвращает описатель (HANDLE) файла. При ошибке INVALID_HANDLE_VALUE. Код ошибки можно получить вызвав функцию GetLastError.

Открытый порт должен быть закрыт перед завершением работы программы. В Win32 закрытие объекта по его описателю выполняет функция CloseHandle:

```
BOOL CloseHandle(
   HANDLE hObject
);
```

Функция имеет единственный параметр - описатель закрываемого объекта. При успешном завершении функция возвращает не нулевое значение, при ошибке нуль

4.5.3. Прием информации и ее накопление в буфере/Передача информации из буфера в интерфейс.

В Python прием и передача данных выполняется функциями read()/write():

```
read(self, size=1) - передается сколько бит нужно считать
write(self, data) - что нужно записать в буфер
```

В Win32 прием и передача данных выполняется функциями ReadFile и WriteFile, то есть теми же самыми, которые используются для работы с дисковыми файлами. Прототипы этих функций выглядят так:

```
BOOL ReadFile(
   HANDLE       hFile,
   LPVOID       lpBuffer,
   DWORD        nNumOfBytesToRead,
   LPDWORD      lpNumOfBytesRead,
   LPOVERLAPPED lpOverlapped
);
BOOL WriteFile(
   HANDLE       hFile,
   LPVOID       lpBuffer,
   DWORD        nNumOfBytesToWrite,
   LPDWORD      lpNumOfBytesWritten,
   LPOVERLAPPED lpOverlapped
);
```

hFile — Описатель открытого файла коммуникационного порта.

lpBuffer — Адрес буфера. Для операции записи данные из этого буфера будут передаваться в порт. Для операции чтения в этот буфер будут помещаться принятые из линии данные.

nNumOfBytesToRead, nNumOfBytesToWrite — Число ожидаемых к приему или предназначеных к передаче байт.

nNumOfBytesRead, nNumOfBytesWritten — Число фактически принятых или переданых байт. Если принято или передано меньше данных, чем запрошено, то для дискового файла это свидетельствует об ошибке, а для коммуникационного порта совсем не обязательно. Причина в тайм-аутах.

lpOverlapped — Адрес структуры OVERLAPPED, используемой для асинхронных операций. Для синхронных операций данный параметр должен быть равным NULL.

4.5.4. Функции управления приемом/передачей COM-порта.

Функция PurgeComm сбрасывает порт. Ее прототип выглядит так:

```
BOOL PurgeComm(
      HANDLE hFile,
      DWORD  dwFlags
   );
```

Вызов этой функции позволяет решить две задачи: очистить очереди приема/передачи в драйвере и завершить все находящиеся в ожидании запросы ввода/вывода. Какие именно действия выполнять, задается вторым параметром (значения можно комбинировать с помощью побитовой операции OR):

PURGE_TXABORT	Немедленно прекращает все операции записи, даже если они не завершены
PURGE_RXABORT	Немедленно прекращает все операции чтения, даже если они не завершены
PURGE_TXCLEAR	Очищает очередь передачи в драйвере
PURGE_RXCLEAR	Очищает очередь приема в драйвере

4.6 Реализация асинхронного обмена информацией.

Синхронный режим обмена довольно редко оказывается подходящим для серьезной работы с внешними устройствами через последовательные порты. Вместо полезной работы

Ваша программа будет ждать завершения ввода/вывода, ведь порты работают значительно медленнее процессора. Да и гораздо лучше отдать время процессора другой программе, чем крутиться в цикле, ожидая какого-либо события. Следовательно, нужно работать в асинхронном режиме с портами.

Начать нужно с событий, связанных с последовательными портами. Нужно указать системе, осуществлять слежение за возникновением связанных с портом событий, устанавливая маску с помощью функции:

```
BOOL SetCommMask(
   HANDLE hFile,
   DWORD  dwEvtMask
);
```

Маска отслеживаемых событий задается вторым параметром. Можно указывать любую комбинацию следующих значений:

EV_BREAK	Состояние разрыва приемной линии
EV_CTS	Изменение состояния линии CTS
EV_DSR	Изменение состояния линии DSR
EV_ERR	Ошибка обрамления, перебега или четности
EV_RING	Входящий звонок на модем (сигнал на линии RI порта)
EV_RLSD	Изменение состояния линии RLSD (DCD)
EV_RXCHAR	Символ принят и помещен в приемный буфер
EV_RXFLAG	Принят символ заданый полем EvtChar структуры DCB использованой для настройки режимов работы порта
EV_TXEMPTY	Из буфера передачи передан последний символ

Если dwEvtMask равно нулю, то отслеживание событий запрещается. Всегда можно получить текущую маску отслеживаемых событий с помощью функции:

```
BOOL GetCommMask(
   HANDLE  hFile,
   LPDWORD lpEvtMask
);
```

Вторым параметром задается адрес переменной, принимающей значение текущей установленной маски отслеживаемых событий. В дополнение к событиям, перечисленным в описании функции SetCommMask, данная функция может возвратить следующие:

EV_EVENT1	Устройство-зависимое событие
EV_EVENT2	Устройство-зависимое событие
EV_PERR	Ошибка принтера
EV_RX80FULL	Приемный буфер заполнен на 80 процентов

Эти дополнительные события используются внутри драйвера. Вы не должны переустанавливать состояние их отслеживания.

Когда маска отслеживаемых событий задана, можно приостановить выполнение своей программы до наступления события. При этом программа не будет занимать процессор. Это выполняется вызовом функции:

```
BOOL WaitCommEvent(
   HANDLE       hFile,
   LPDWORD      lpEvtMask,
   LPOVERLAPPED lpOverlapped,
);
```

Замечу, что в переменной, адресуемой вторым параметром, не будут устанавливаться внутренние события драйвера (перечислены в описании функции GetCommMask). В единичное состояние установятся только те биты, которые соответствуют реально произошедшим событиям.

Адрес структуры OVERLAPPED требуется для асинхронного ожидания (возможно и такое). Замечу только, что при асинхронном ожидании данная функция может завершиться с ошибкой, если в процессе этого ожидания будет вызвана функция SetCommMask для переустановки маски событий. Кроме того, связанное со структурой OVERLAPPED событие (объект создаваемый функцией CreateEvent, а не событие порта) должно быть с ручным сбросом. Вообще, поведение функции с ненулевым указателем на структуру OVERLAPPED аналогично поведению функций чтения и записи.

Описание функции CreateEvent выглядит так:

```
HANDLE CreateEvent(
  LPSECURITY_ATTRIBUTES lpEventAttributes,
  BOOL bManualReset,
  BOOL bInitialState,
  LPCTSTR lpName
 );
```

Опишем кратко параметры этой функции:

lpEventAttributes — указатель на структуру SECURITY_ATTRIBUTES, которая определяет может ли дочерний процесс унаследовать возвращаемый описатель. Если lpEventAttributes — NULL, описатель не может быть унаследован.

bManualReset — определяет является ли создаваемый объект события сбрасываемым вручную или автоматически. Если TRUE, вы должны использовать функцию ResetEvent для ручного сброса в несигнальное состояние. Если FALSE, Windows автоматически сбросит в несигнальное состояние после того как ожидающий поток возобновит выполнение.

bInitialState — Определяет начальное состояние объекта события. Если TRUE, начальное состояние сигнальное, иначе — несигнальное.

pName — указывает на сточку, определяющую имя объекта события specifying the name of the event object. Если lpName — NULL, объект события будет создан без имени.

5. Канальный уровень.

5.1 Функции канального уровня.

На канальном уровне выполняются следующие функции:

1. Запрос логического соединения;
2. Разбивка данных на блоки (кадры);

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

5.2 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.

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 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.

Рассмотрим алгоритм кода Хемминга:

Идея кодов Хемминга заключается в разбиении данных на блоки фиксированной длины и вводе в эти блоки 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:

$$2^m * (n+1)=2^n$$

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 длинной 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 к – 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 $9 = 2^0 + 2^3 = 1 + 8$.

5.4. Процедуры взаимодействия.

5.4.1. Успешное установление логического соединения.

На прикладном уровне необходимо нажать на кнопку «Открыть порт». После этого на канальный уровень подается команда послать UPLINK-кадр. На физическом уровне передаются биты. Второй компьютер на канальном уровне получает UPLINK-кадр. На прикладном уровне отображается уведомление о соединении. Второй компьютер на канальном уровне передает ACK_UPLINK-кадр, по физическому уровню отправляются биты. Первый компьютер на канальном уровне получает ACK_UPLINK-кадр. На прикладном уровне, в окне отображается сообщение об успешном установлении соединения.

5.4.2. Невозможность установления логического соединения.

На прикладном уровне необходимо нажать на кнопку «Открыть порт». После этого на канальный уровень подается команда послать UPLINK-кадр. На физическом уровне передаются биты. Из-за проблем на физическом уровне, биты не достигают второго компьютера. По истечении таймаута, канальный уровень передает, что ACK_UPLINK кадр не пришел. На прикладном уровне, в окне появляется сообщение о невозможности установления соединения.

5.4.3. Успешная передача сообщения.

На прикладном уровне необходимо ввести сообщение в главном окне и нажать «Отправить». После этого на канальный уровень подается команда послать DAT-кадр с сообщением. На физическом уровне передаются биты. Второй компьютер на канальном уровне получает DAT-кадр. На прикладном уровне отображается в главном окне сообщение от первого компьютера. Второй компьютер на канальном уровне передает ACK-кадр об успешном принятии сообщения, по физическому уровню отправляются биты. Первый

компьютер на канальном уровне получает ACK-кадр. На прикладном уровне, в окне исходящих сообщений отображается сообщение.

5.4.4. Поддержание логического соединения.

Канальный уровень передает LINKACTIVE-кадр. На физическом уровне передаются биты. Второй компьютер на канальном уровне получает LINKACTIVE-кадр. Второй компьютер на канальном уровне передает ACK_ LINKACTIVE-кадр, по физическому уровню отправляются биты. Первый компьютер на канальном уровне получает ACK_ LINKACTIVE-кадр об успешности логического соединения. Если LINKACTIVE-кадр не доходит до второго компьютера, то по истечении таймаута на прикладной уровень в окне сообщений приходит уведомление о недоступности канала.

5.4.5. Передача сообщения с ошибкой. Потеря сообщения.

На прикладном уровне необходимо ввести сообщение в главном окне и нажать «Отправить». После этого на канальный уровень подается команда послать DAT-кадр с сообщением. На физическом уровне передаются биты, но из-за внешнего воздействия происходит ошибка. Второй компьютер на канальном уровне получает DAT-кадр и замечает ошибку в битах. Второй компьютер на канальном уровне передает RET-кадр об ошибке сообщения, по физическому уровню отправляются биты. Первый компьютер на канальном уровне получает RET-кадр. Канальный уровень повторно передает DAT-кадр, но биты не достигают второго компьютера. По истечении таймаута, на прикладном уровне отображается уведомление о недоставке сообщения.

5.4.6. Успешное разъединение логического соединения.

На прикладном уровне необходимо нажать на кнопку «Закрыть порт». После этого на канальный уровень подается команда послать DOWNLINK-кадр. На физическом уровне передаются биты. Второй компьютер на канальном уровне получает DOWNLINK-кадр о запросе разъединения. На прикладном уровне отображается уведомление о разъединении. Второй компьютер на канальном уровне передает ACK_ DOWNLINK-кадр, по физическому уровню отправляются биты. Первый компьютер на канальном уровне получает ACK_ DOWNLINK-кадр о подтверждении разъединения. На прикладном уровне, в окне отображается сообщение о подтверждении разъединения.

5.4.7. Невозможность разъединения логического соединения.

На прикладном уровне необходимо нажать на кнопку «Закрыть порт». После этого на канальный уровень подается команда послать DOWNLINK-кадр. На физическом уровне передаются биты. Из-за проблем на физическом уровне, биты не достигают второго компьютера. По истечении таймаута на прикладном уровне в окне отображается уведомление о невозможности разъединения логического соединения.

5.5. Формат кадров.

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

5.5.1 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 сообщения(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

5.5.2 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

5.5.3 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 циклическим кодом пользовательских сообщений. Формат эти кадров:

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

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

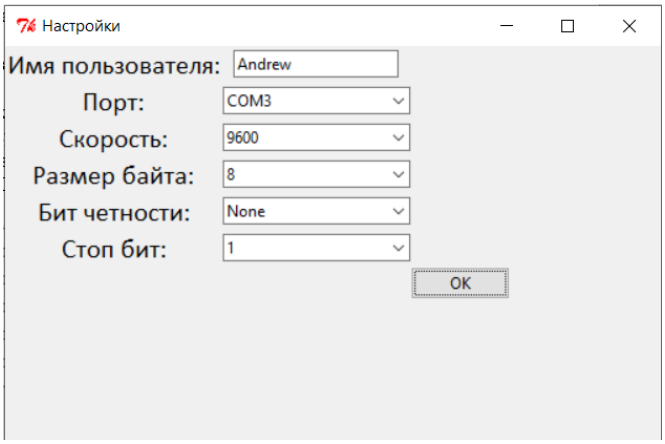
6. Прикладной уровень.

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 на Python. При его разработке учитывались рекомендации по простоте, удобству и функциональности интерфейса.

При запуске программы появляется форма, в которой необходимо ввести имя собеседника и выбрать параметры COM-порта.



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

- Набор и редактирование сообщений
- Отображение текущей истории
- Открытие/закрытие выбранного последовательного порта
- Исходящие/приходящие сообщения
- Мониторинг активности соединения
- Использование меню

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