

Утверждаю:

Галкин В.А. "___" _____ 2020 г.

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

Описание программы

(вид документа)

писчая бумага

(вид носителя)

27

(количество листов)

ИСПОЛНИТЕЛИ:

студенты группы ИУ5-61Б

Матиенко А.П. _____

Молева А.А. _____

Белоусов Е.А. _____

1. Общие сведения

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

Программа выполняется на языке программирования Python/С и работает под управлением операционной системы Windows XP и выше.

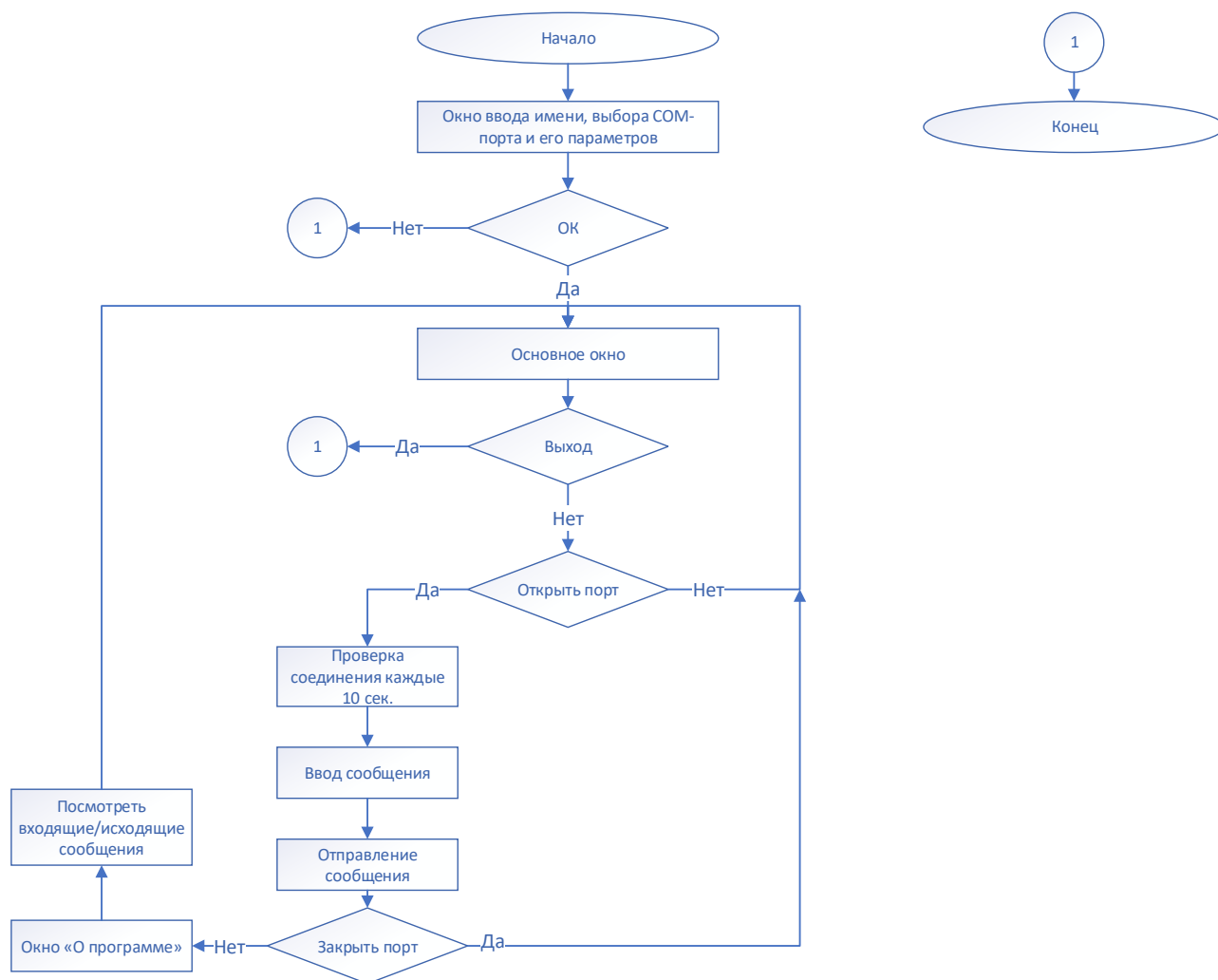
2. Назначение разработки

Программа должна реализовывать функцию передачи текстовых сообщений и файлов между двумя ПЭВМ, соединенными через интерфейс RS-232C с использованием нуль-модемного кабеля.

3. Описание логической структуры

3.1. Алгоритм интерфейсной части программы

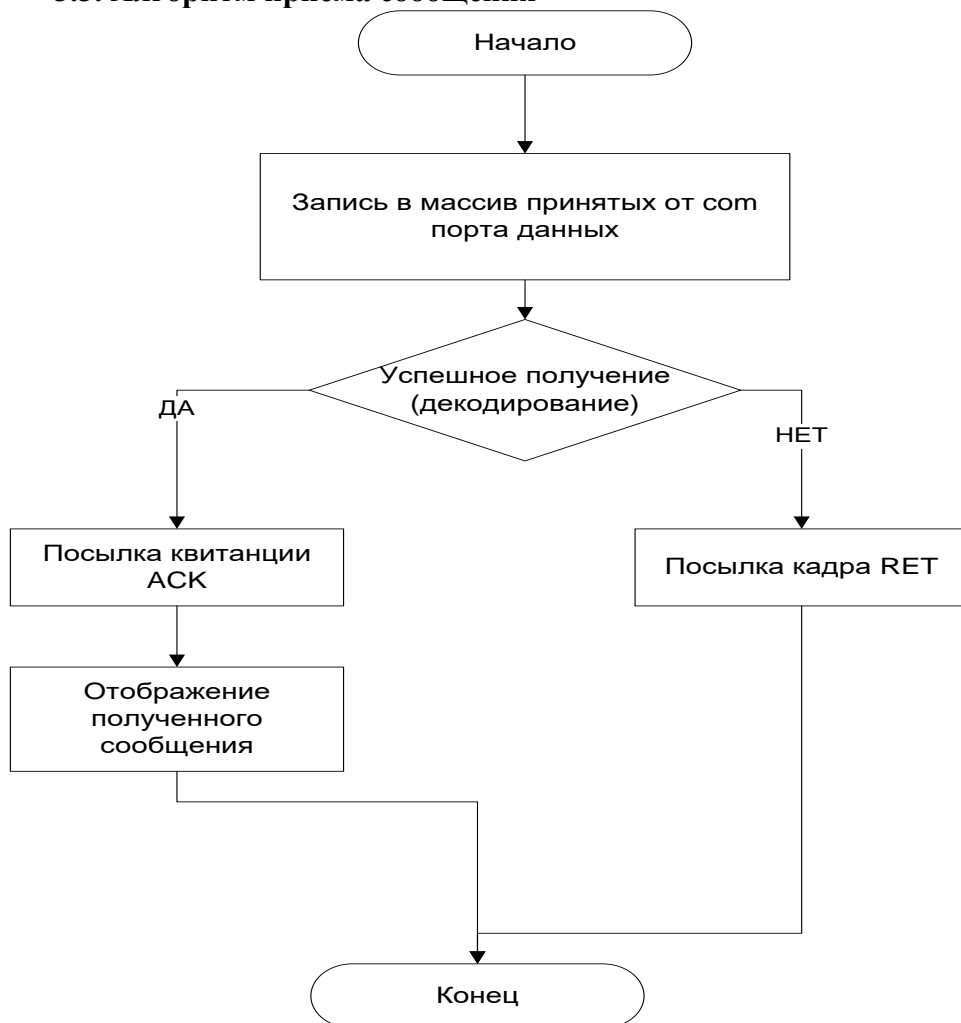
Алгоритм интерфейсной части приведен на рисунке.



3.2. Алгоритм передачи сообщения



3.3. Алгоритм приема сообщения



4. Используемые технические средства

Программа должна работать на IBM-совместимой ЭВМ следующей конфигурации:

- 4.1. Центральный процессор Pentium 4 или выше;
- 4.2. Объем оперативной памяти 512 Мб;
- 4.3. Видеоадаптер и монитор VGA и выше;
- 4.4. Стандартная клавиатура;
- 4.5. 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, выбранный пользователем.

5.2. Выходные данные.

Выходными данными являются:

- текст переданного сообщения на ПЭВМ;
- 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

7. Спецификация функций

```
def main(): - головная функция программы
def configure_window(ser): - окно настроек
def clicked(): - нажатие на кнопку «OK»
def validation(name, com_port, speed_b, size_b, parity_b, bit_stop, ser): - Валидация
параметров COM-порта
def cut_port_name(str): - Обрезаем полное имя COM-порта до <COM(цифра)>
def chat(ser): - главное окно программы
def check_connect(): - проверка соединения, посылает сигнал «ACK_LINKACTIVE»
def fn_in(): - функция приема строки
def fn_out(): - запуск основного потока
def fn_send(): - отправление сообщения
def fn_disp(): - отображение сообщения на дисплее
def open_port(): - кнопка «Открыть порт»
def about_program(): - меню
def source_message(): - Окно - Отправленные сообщения
def dest_message(): - Окно - Пришедшие сообщения
```

7.1. Функции в классе SerialBase

Инициализация

```
def port(self, port): - порт
def baudrate(self, baudrate): - пропускная способность
def bytesize(self, bytesize): - размер байта
def parity(self, parity): - бит четности
def stopbits(self, stopbits): - стопбит
def timeout(self, timeout): - таймаут
Настройки:
def write_timeout(self, timeout): - изменить таймаут
def xonxoff(self, xonxoff): - XON/XOFF
def rtscts(self, rtscts): - Change RTS/CTS flow control setting.
def dsrdtr(self, dsrdtr=None): - Change DsrDtr flow control setting.
def inter_byte_timeout(self, ic_timeout): - Change inter-byte timeout setting.
```

```
def __repr__(self): - Отобразить всю информацию о порте
```

7.2. Функции в классе Serial

```
def open(self): - открытие порта
```

```
def _reconfigure_port(self): - Set communication parameters on opened port.(настроить порт)
```

```
def close(self): - закрытие порта
```

```
def _cancel_overlapped_io(self, overlapped): - прекращение чтения/записи данных
```

```
def cancel_read(self): - ссылается на _cancel_overlapped_io
```

```
def cancel_write(self): - ссылается на _cancel_overlapped_io
```

```
def ft_write(self, data): - запись в буфер
```

```
def in_waiting(self): - возвращает количество байт в input буфере
```

```
def ft_read(self, size=1): - чтение из буфера
```

8. Листинг основных функций

main.py:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from my_package.ft_serial_1 import Serial
from my_package.conf_com_port import configure_window
from my_package.chat import chat
```

```
##---Fox exe
```

```
if 0:
    import UserList
    import UserString
    import UserDict
    import itertools
    import collections
    import future.backports.misc
    import commands
    import base64
    import __builtin__
    import math
    import reprlib
    import functools
    import re
    import subprocess
```

```
###
```

```
def main():
    ser = Serial()
    ok_button = configure_window(ser)
    # ser.timeout = 2
    if ok_button:
        chat(ser)
```

```
if __name__ == "__main__":
    main()
```

Configurations.py:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
"""
```

```
COM-порт: Параметры
```

```
"""
```

```
BAUDRATES = (50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800,
              9600, 19200, 38400, 57600, 115200, 230400, 460800, 500000,
              576000, 921600, 1000000, 1152000, 1500000, 2000000, 2500000,
              3000000, 3500000, 4000000)
PARITY_NONE, PARITY_EVEN, PARITY_ODD, PARITY_MARK, PARITY_SPACE = 'None',
```

```

'Even', 'Odd', 'Mark', 'Space'
STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO = (1, 1.5, 2)

PARITY_NAMES = {
    PARITY_NONE: 'None',
    PARITY_EVEN: 'Even',
    PARITY_ODD: 'Odd',
    PARITY_MARK: 'Mark',
    PARITY_SPACE: 'Space',
}

FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS = (5, 6, 7, 8)
BYTESIZES = (FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS)
PARITIES = (PARITY_NONE, PARITY_EVEN, PARITY_ODD, PARITY_MARK, PARITY_SPACE)
STOPBITS = (STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO)

```

ft_serial 1.py:

```

import ctypes
import time
from serial import win32
from .code_Hemming import *
from .ft_serial import SerialBase, to_bytes
from . import ft_serial

class Serial(SerialBase):
    BAUDRATES = (50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800,
                 2400, 4800, 9600, 19200, 38400, 57600, 115200)

    def __init__(self, *args, **kwargs):
        self._port_handle = None
        self._overlapped_read = None
        self._overlapped_write = None
        super(Serial, self).__init__(*args, **kwargs)

    """
    Open port with current settings
    """

    def open(self):
        if self._port is None:
            print("ERROR: Port must be configured before it can be used.")
            exit(1)
        if self.is_open:
            print("ERROR: Port is already opened.")
            exit(1)
        port = self.name
        try:
            if port.upper().startswith('COM') and int(port[3:]) > 8:
                port = '\\\\.\\' + port
        except ValueError:
            pass
        self._port_handle = win32.CreateFile(
            port,
            win32.GENERIC_READ | win32.GENERIC_WRITE,
            0, # exclusive access
            None, # no security
            win32.OPEN_EXISTING,
            win32.FILE_ATTRIBUTE_NORMAL | win32.FILE_FLAG_OVERLAPPED,
            0
        )
        """
        Bad COM port
        """

```

```

if self._port_handle == win32.INVALID_HANDLE_VALUE:
    self._port_handle = None
    print("ERROR: Could not open port {}".format(self.port))
    exit(1)

try:
    self._overlapped_read = win32.OVERLAPPED()
    self._overlapped_read.hEvent = win32.CreateEvent(None, 1, 0, None)
    self._overlapped_write = win32.OVERLAPPED()
    self._overlapped_write.hEvent = win32.CreateEvent(None, 0, 0, None)

    # Setup a 4k buffer
    win32.SetupComm(self._port_handle, 4096, 4096)

    # Save original timeout values:
    self._orgTimeouts = win32.COMMTIMEOUTS()
    win32.GetCommTimeouts(self._port_handle,
        ctypes.byref(self._orgTimeouts))

    self._reconfigure_port()

    win32.PurgeComm(
        self._port_handle,
        win32.PURGE_TXCLEAR | win32.PURGE_TXABORT |
        win32.PURGE_RXCLEAR | win32.PURGE_RXABORT)
except:
    try:
        self._close()
    except:
        # ignore any exception when closing the port
        # also to keep original exception that happened when setting up
        pass
    self._port_handle = None
    raise
else:
    self.is_open = True

def _reconfigure_port(self):
    """Set communication parameters on opened port."""
    if not self._port_handle:
        print("ERROR: Can only operate on a valid port handle")
        exit(1)

    timeouts = win32.COMMTIMEOUTS()
    if self._timeout is None:
        pass
    elif self._timeout == 0:
        timeouts.ReadIntervalTimeout = win32.MAXDWORD
    else:
        timeouts.ReadTotalTimeoutConstant = max(int(self._timeout * 1000), 1)
    if self._timeout != 0 and self._inter_byte_timeout is not None:
        timeouts.ReadIntervalTimeout = max(int(self._inter_byte_timeout *
1000), 1)

    if self._write_timeout is None:
        pass
    elif self._write_timeout == 0:
        timeouts.WriteTotalTimeoutConstant = win32.MAXDWORD
    else:
        timeouts.WriteTotalTimeoutConstant = max(int(self._write_timeout *
1000), 1)

    win32.SetCommTimeouts(self._port_handle, ctypes.byref(timeouts))
    win32.SetCommMask(self._port_handle, win32.EV_ERR)

```



```

"""Setup the connection info
    Get state and modify it"""
comDCB = win32.DCB()
win32.GetCommState(self._port_handle, ctypes.byref(comDCB))
"""Set baudrate"""
comDCB.BaudRate = self._baudrate
"""Set bytesize"""
if self._bytesize == ft_serial.FIVEBITS:
    comDCB.ByteSize = 5
elif self._bytesize == ft_serial.SIXBITS:
    comDCB.ByteSize = 6
elif self._bytesize == ft_serial.SEVENBITS:
    comDCB.ByteSize = 7
elif self._bytesize == ft_serial.EIGHTBITS:
    comDCB.ByteSize = 8

"""Set parity"""
if self._parity == ft_serial.PARITY_NONE:
    comDCB.Parity = win32.NOPARITY
    comDCB.fParity = 0
elif self._parity == ft_serial.PARITY_EVEN:
    comDCB.Parity = win32.EVENPARITY
    comDCB.fParity = 1 # Enable Parity Check
elif self._parity == ft_serial.PARITY_ODD:
    comDCB.Parity = win32.ODDPARITY
    comDCB.fParity = 1 # Enable Parity Check
elif self._parity == ft_serial.PARITY_MARK:
    comDCB.Parity = win32.MARKPARITY
    comDCB.fParity = 1 # Enable Parity Check
elif self._parity == ft_serial.PARITY_SPACE:
    comDCB.Parity = win32.SPACEPARITY
    comDCB.fParity = 1 # Enable Parity Check
else:
    print("ERROR: Unsupported parity mode: {}".format(self._parity))
    exit(1)

"""Set stopbit"""
if self._stopbits == ft_serial.STOPBITS_ONE:
    comDCB.StopBits = win32.ONESTOPBIT
elif self._stopbits == ft_serial.STOPBITS_ONE_POINT_FIVE:
    comDCB.StopBits = win32.ONE5STOPBITS
elif self._stopbits == ft_serial.STOPBITS_TWO:
    comDCB.StopBits = win32.TWOSTOPBITS
else:
    print("ERROR: Unsupported number of stop bits:
    {}".format(self._stopbits))
    exit(1)

comDCB.fBinary = 1 # Enable Binary Transmission
# Char. w/ Parity-Err are replaced with 0xff (if fErrorChar is set to
TRUE)
if self._rs485_mode is None:
    if self._rtscts:
        comDCB.fRtsControl = win32.RTS_CONTROL_HANDSHAKE
    else:
        comDCB.fRtsControl = win32.RTS_CONTROL_ENABLE if self._rts_state
else win32.RTS_CONTROL_DISABLE
    comDCB.fOutxCtsFlow = self._rtscts

if self._dsrdtr:
    comDCB.fDtrControl = win32.DTR_CONTROL_HANDSHAKE
else:
    comDCB.fDtrControl = win32.DTR_CONTROL_ENABLE if self._dtr_state else

```

```

win32.DTR_CONTROL_DISABLE
comDCB.fOutxDsrFlow = self._dsrdtr
comDCB.fOutX = self._xonxoff
comDCB.fInX = self._xonxoff
comDCB.fNull = 0
comDCB.fErrorChar = 0
comDCB.fAbortOnError = 0
comDCB.XonChar = ft_serial.XON
comDCB.XoffChar = ft_serial.XOFF

if not win32.SetCommState(self._port_handle, ctypes.byref(comDCB)):
    print(
        'ERROR: Cannot configure port, something went wrong. '
        'Original message: {!r}'.format(ctypes.WinError()))
    exit(1)

"""Close port"""
def close(self):
    if self.is_open:
        self._close()
        self.is_open = False

def _close(self):
    if self._port_handle is not None:
        win32.SetCommTimeouts(self._port_handle, self._orgTimeouts)
        if self._overlapped_read is not None:
            self.cancel_read()
            win32.CloseHandle(self._overlapped_read.hEvent)
            self._overlapped_read = None
        if self._overlapped_write is not None:
            self.cancel_write()
            win32.CloseHandle(self._overlapped_write.hEvent)
            self._overlapped_write = None
        win32.CloseHandle(self._port_handle)
        self._port_handle = None

"""##-----Stop read information-----##"""
def _cancel_overlapped_io(self, overlapped):
    """Cancel a blocking read operation, may be called from other thread"""
    # check if read operation is pending
    rc = win32.DWORD()
    err = win32.GetOverlappedResult(
        self._port_handle,
        ctypes.byref(overlapped),
        ctypes.byref(rc),
        False)
    if not err and win32.GetLastError() in (win32.ERROR_IO_PENDING,
win32.ERROR_IO_INCOMPLETE):
        # cancel, ignoring any errors (e.g. it may just have finished on its
own)
        win32.CancelIoEx(self._port_handle, overlapped)

def cancel_read(self):
    self._cancel_overlapped_io(self._overlapped_read)

"""##-----Stop write information-----##"""
def cancel_write(self):
    self._cancel_overlapped_io(self._overlapped_write)

"""-----Write info-----"""

def ft_write(self, data):
    if not self.is_open:

```

```

        print("Port is not opened")
        exit(1)
    data_encode = encode(data)
    data_encode_with_errors = set_errors(data_encode)
    data_encode_with_errors = data_encode_with_errors.encode('utf-8')
    n = win32.DWORD()
    success = win32.WriteFile(self._port_handle, data_encode_with_errors,
len(data_encode_with_errors),
                                ctypes.byref(n), self._overlapped_write)
    self._buffer.append(data_encode_with_errors)
    return len(data)

def write(self, data):
    if not self.is_open:
        print("Port is not opened")
        exit(1)
    data = to_bytes(data)
    if data:
        n = win32.DWORD()
        success = win32.WriteFile(self._port_handle, data, len(data),
                                ctypes.byref(n), self._overlapped_write)

        if self._write_timeout != 0:
            if not success and win32.GetLastError() not in (win32.ERROR_SUCCESS,
win32.ERROR_IO_PENDING):
                print("WriteFile failed ({!r})".format(ctypes.WinError()))
                exit(1)
            win32.GetOverlappedResult(self._port_handle, self._overlapped_write,
                                ctypes.byref(n), True)

            if win32.GetLastError() == win32.ERROR_OPERATION_ABORTED:
                return n.value
            if n.value != len(data):
                print("Write timeout")
                exit(1)
            return n.value
        else:
            errorcode = win32.ERROR_SUCCESS if success else win32.GetLastError()
            if errorcode in (win32.ERROR_INVALID_USER_BUFFER,
win32.ERROR_NOT_ENOUGH_MEMORY,
                                win32.ERROR_OPERATION_ABORTED):
                return 0
            elif errorcode in (win32.ERROR_SUCCESS, win32.ERROR_IO_PENDING):
                # no info on true length provided by OS function in async mode
                return len(data)
            else:
                print("WriteFile failed ({!r})".format(ctypes.WinError()))
                exit(1)
    else:
        return 0

@property
def in_waiting(self):
    """Return the number of bytes currently in the input buffer."""
    flags = win32.DWORD()
    comstat = win32.COMSTAT()
    if not win32.ClearCommError(self._port_handle, ctypes.byref(flags),
ctypes.byref(comstat)):
        # print("ClearCommError failed ({!r})".format(ctypes.WinError()))
        pass
    return comstat.cbInQue

"""-----Read info-----"""

def ft_read(self, size=1):

```

```

    if not self.is_open:
        print("ERROR: Port is not opened")
    if size > 0:
        win32.ResetEvent(self._overlapped_read.hEvent)
        flags = win32.DWORD()
        comstat = win32.COMSTAT()
        n = min(comstat.cbInQue, size) if self.timeout == 0 else size
        if n > 0:
            buf = ctypes.create_string_buffer(n)
            rc = win32.DWORD()
            read_ok = win32.ReadFile(self._port_handle,
                                    buf,
                                    n,
                                    ctypes.byref(rc),
                                    ctypes.byref(self._overlapped_read))
            buffer = buf.raw.decode('utf-8')
            buffer = decode(buffer)
            return buffer
        else:
            return []

def read(self, size=1):
    if not self.is_open:
        print("ERROR: Port is not opened")
        exit(1)
    if size > 0:
        win32.ResetEvent(self._overlapped_read.hEvent)
        flags = win32.DWORD()
        comstat = win32.COMSTAT()
        if not win32.ClearCommError(self._port_handle, ctypes.byref(flags),
        ctypes.byref(comstat)):
            print("ERROR: ClearCommError failed
            (!r})).format(ctypes.WinError()))
            exit(1)
        n = min(comstat.cbInQue, size) if self.timeout == 0 else size
        if n > 0:
            buf = ctypes.create_string_buffer(n)
            rc = win32.DWORD()
            read_ok = win32.ReadFile(self._port_handle,
                                    buf,
                                    n,
                                    ctypes.byref(rc),
                                    ctypes.byref(self._overlapped_read))
            if not read_ok and win32.GetLastError() not in (win32.ERROR_SUCCESS,
            win32.ERROR_IO_PENDING):
                print("ERROR: ReadFile failed (!r})).format(ctypes.WinError()))
                exit(1)
            if not read_ok:
                print("ERROR: Something bad")
                return buf.value
            result_ok = win32.GetOverlappedResult(self._port_handle,
            ctypes.byref(self._overlapped_read),
            ctypes.byref(rc),
            True)
            if not result_ok:
                if win32.GetLastError() != win32.ERROR_OPERATION_ABORTED:
                    raise SerialException("GetOverlappedResult failed
                    (!r})).format(ctypes.WinError()))
                read = buf.raw[:rc.value]
            else:
                read = bytes()
        else:

```

```

        read = bytes()
        return bytes(read)

ft_serial.py:

#!/usr/bin/env python
# -*- coding: utf-8 -*-
import io
import time
import sys

PARITY_NONE, PARITY_EVEN, PARITY_ODD, PARITY_MARK, PARITY_SPACE = 'None',
'Even', \
                                                                    'Odd', 'Mark',
'Space'
STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO = (1, 1.5, 2)
FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS = (5, 6, 7, 8)

PARITY_NAMES = {
    PARITY_NONE: 'None',
    PARITY_EVEN: 'Even',
    PARITY_ODD: 'Odd',
    PARITY_MARK: 'Mark',
    PARITY_SPACE: 'Space',
}

def to_bytes(seq):
    """convert a sequence to a bytes type"""
    if isinstance(seq, bytes):
        return seq
    elif isinstance(seq, bytearray):
        return bytes(seq)
    elif isinstance(seq, memoryview):
        return seq.tobytes()
    elif isinstance(seq, str):
        raise TypeError('unicode strings are not supported, please encode to
bytes: {!r}'.format(seq))
    else:
        # handle list of integers and bytes (one or more items) for Python 2 and
3
        return bytes(bytearray(seq))

# create control bytes
XON = to_bytes([17])
XOFF = to_bytes([19])

CR = to_bytes([13])
LF = to_bytes([10])

class SerialBase(io.RawIOBase):
    """\
    Serial port base class. Provides __init__ function and properties to
    get/set port settings.
    """

    # default values, may be overridden in subclasses that do not support all
    values
    BAUDRATES = (50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800,
                  9600, 19200, 38400, 57600, 115200, 230400, 460800, 500000,
                  576000, 921600, 1000000, 1152000, 1500000, 2000000, 2500000,
                  3000000, 3500000, 4000000)
    BYTESIZES = (FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS)
    PARITIES = (PARITY_NONE, PARITY_EVEN, PARITY_ODD, PARITY_MARK, PARITY_SPACE)
    STOPBITS = (STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO)

```

```

def __init__(self,
              port=None,
              baudrate=9600,
              bytesize=EIGHTBITS,
              parity=PARITY_NONE,
              stopbits=STOPBITS_ONE,
              timeout=None,
              xonxoff=False,
              rtscts=False,
              write_timeout=None,
              dsrdtr=False,
              inter_byte_timeout=None,
              username=None,
              **kwargs):
    """Initialize comm port object. If a "port" is given, then the port will
    be opened immediately. Otherwise a Serial port object in closed state
    is returned.
    """

    self.is_open = False
    self.portstr = None
    # correct values are assigned below through properties
    self.name = None
    self._port = None
    self._baudrate = None
    self._bytesize = None
    self._parity = None
    self._stopbits = None
    self._timeout = None
    self._write_timeout = None
    self._xonxoff = None
    self._rtscts = None
    self._dsrdtr = None
    self._inter_byte_timeout = None
    self._rs485_mode = None # disabled by default
    self._rts_state = True
    self._dtr_state = True
    self._break_state = False
    self._exclusive = None
    self._buffer = []
    self._username = None

    # assign values using get/set methods using the properties feature
    self.port = port
    self.baudrate = baudrate
    self.bytesize = bytesize
    self.parity = parity
    self.stopbits = stopbits
    self.timeout = timeout
    self.write_timeout = write_timeout
    self.xonxoff = xonxoff
    self.rtscts = rtscts
    self.dsrdtr = dsrdtr
    self.inter_byte_timeout = inter_byte_timeout

    ##-- Открываем порт
    if port is not None:
        self.open()

    ##-----Порт-----##
    @property
    def port(self):

```

```

        """\
        Get the current port setting. The value that was passed on init
or using        setPort() is passed back.
        """
        return self._port

@port.setter
def port(self, port):
    if port is not None and not isinstance(port, str):
        print("ERROR: \"port\" must be None or a string")
        exit(1)
    was_open = self.is_open
    if was_open:
        self.close()
    self.portstr = port
    self._port = port
    self.name = self.portstr
    if was_open:
        self.open()

##-----Скорость-----##
@property
def baudrate(self):
    return self._baudrate

@baudrate.setter
def baudrate(self, baudrate):
    try:
        b = int(baudrate)
    except TypeError:
        raise ValueError("Not a valid baudrate: {!r}".format(baudrate))
    else:
        if b < 0:
            print("ERROR: \"baudrate\" must be positive")
            exit(1)
        self._baudrate = b
        if self.is_open:
            pass

##_____Бит данных_____##
@property
def bytesize(self):
    """Get the current byte size setting."""
    return self._bytesize

@bytesize.setter
def bytesize(self, bytesize):
    """Change byte size."""
    if bytesize not in self.BYTESIZES:
        print("ERROR: Not a valid byte size: \"{}\" + str(bytesize) + "\"")
        exit(1)
    self._bytesize = bytesize
    if self.is_open:
        pass
    # self._reconfigure_port()

##_____Бит четности_____##
@property
def parity(self):
    """Get the current parity setting."""
    return self._parity

@parity.setter

```

```

def parity(self, parity):
    """Change parity setting."""
    if parity not in self.PARITIES:
        print("ERROR: Not a valid parity: {!r}".format(parity))
        exit(1)
    self._parity = parity
    if self.is_open:
        pass
        # self._reconfigure_port()

##-----Стопбит-----##
@property
def stopbits(self):
    """Get the current stop bits setting."""
    return self._stopbits

@stopbits.setter
def stopbits(self, stopbits):
    """Change stop bits size."""
    if stopbits not in self.STOPBITS:
        print("ERROR: Not a valid stop bit size: {!r}".format(stopbits))
        exit(1)
    self._stopbits = stopbits
    if self.is_open:
        pass
        # self._reconfigure_port()

##-----Set timeout-----##
@property
def timeout(self):
    return self._timeout

@timeout.setter
def timeout(self, timeout):
    if timeout is not None:
        try:
            timeout + 1
        except TypeError:
            raise ValueError("Not a valid timeout: {!r}".format(timeout))
        if timeout < 0:
            raise ValueError("Not a valid timeout: {!r}".format(timeout))
    self._timeout = timeout
    if self.is_open:
        self._reconfigure_port()

##-----Set the copy of timeout-----##
@property
def write_timeout(self):
    """Get the current timeout setting."""
    return self._write_timeout

@write_timeout.setter
def write_timeout(self, timeout):
    """Change timeout setting."""
    if timeout is not None:
        if timeout < 0:
            raise ValueError("Not a valid timeout: {!r}".format(timeout))
        try:
            timeout + 1          # test if it's a number, will throw a TypeError
        except TypeError:
            raise ValueError("Not a valid timeout: {!r}".format(timeout))

    self._write_timeout = timeout

```

if not...


```

        if self.is_open:
            self._reconfigure_port()

##-----Set xonxoff-----##

@property
def xonxoff(self):
    """Get the current XON/XOFF setting."""
    return self._xonxoff

@xonxoff.setter
def xonxoff(self, xonxoff):
    """Change XON/XOFF setting."""
    self._xonxoff = xonxoff
    if self.is_open:
        self._reconfigure_port()

##-----Set rtscts-----##

@property
def rtscts(self):
    """Get the current RTS/CTS flow control setting."""
    return self._rtscts

@rtscts.setter
def rtscts(self, rtscts):
    """Change RTS/CTS flow control setting."""
    self._rtscts = rtscts
    if self.is_open:
        self._reconfigure_port()

##-----Set dsrdtr-----##

@property
def dsrdtr(self):
    """Get the current DSR/DTR flow control setting."""
    return self._dsrdtr

@dsrdtr.setter
def dsrdtr(self, dsrdtr=None):
    """Change DsrDtr flow control setting."""
    if dsrdtr is None:
        # if not set, keep backwards compatibility and follow rtscts setting
        self._dsrdtr = self._rtscts
    else:
        # if defined independently, follow its value
        self._dsrdtr = dsrdtr
    if self.is_open:
        self._reconfigure_port()

##-----Set inter byte timeout-----##

@property
def inter_byte_timeout(self):
    """Get the current inter-character timeout setting."""
    return self._inter_byte_timeout

@inter_byte_timeout.setter
def inter_byte_timeout(self, ic_timeout):
    """Change inter-byte timeout setting."""
    if ic_timeout is not None:
        if ic_timeout < 0:
            raise ValueError("Not a valid timeout: {!r}".format(ic_timeout))
        try:

```

```

        ic_timeout + 1      # test if it's a number, will throw a
TypeError if not...
    except TypeError:
        raise ValueError("Not a valid timeout: {!r}".format(ic_timeout))

    self._inter_byte_timeout = ic_timeout
    if self.is_open:
        self._reconfigure_port()

    ##-----Display all info about port-----##
    def __repr__(self):
        """String representation of the current port settings and its state."""
        return '{name}<id=0x{id:x}, open={p.is_open}>(port={p.portstr!r}, ' \
            'baudrate={p.baudrate!r}, bytesize={p.bytesize!r},
parity={p.parity!r}, ' \
            'stopbits={p.stopbits!r}, timeout={p.timeout!r},
xonxoff={p.xonxoff!r}, ' \
            'rtscts={p.rtscts!r}, dsrdtr={p.dsrdtr!r})'.format(
            name=self.__class__.__name__, id=id(self), p=self)

```

conf_com_port.py:

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
from tkinter import *
from tkinter.ttk import *
from my_package.configurations import BAUDRATES, BYTESIZES, PARITIES, STOPBITS
import serial
from serial.tools import list_ports
from my_package.validation import validation, cut_port_name

def configure_window(ser):
    """Создание окна настроек параметров"""
    conf_window = Tk()
    conf_window.geometry('500x300')
    conf_window.title('Настройки')

    """Имя пользователя"""
    label_name = Label(conf_window, text='Имя пользователя:', font=("Calibri",
15))
    label_name.grid(row=0, column=0)
    default_name = StringVar(conf_window, value='Andrew')
    name = Entry(conf_window, width=20, textvariable=default_name)
    name.grid(row=0, column=1)

    """COM-порт"""
    label_port = Label(conf_window, text='Порт:', font=("Calibri", 15))
    label_port.grid(row=1, column=0)
    com_port = Combobox(conf_window)
    com_port['values'] = cut_port_name(list_ports.comports())
    com_port.current(0)
    com_port.grid(row=1, column=1)

    """Скорость обмена"""
    label_speed = Label(conf_window, text='Скорость:', font=("Calibri", 15))
    label_speed.grid(row=2, column=0)
    speed_b = Combobox(conf_window)
    speed_b['values'] = BAUDRATES
    speed_b.current(12)
    speed_b.grid(row=2, column=1)

    """Размер байта"""
    label_byte_size = Label(conf_window, text='Размер байта:', font=("Calibri",

```

```

15))
    label_byte_size.grid(row=3, column=0)
    size_b = Combobox(conf_window)
    size_b['values'] = BYTESIZES
    size_b.current(3)
    size_b.grid(row=3, column=1)

    """Бит четности"""
    label_bit_parity = Label(conf_window, text='Бит четности:', font=("Calibri",
15))
    label_bit_parity.grid(row=4, column=0)
    parity_b = Combobox(conf_window)
    parity_b['values'] = PARITIES
    parity_b.current(0)
    parity_b.grid(row=4, column=1)

    """Стоп бит"""
    label_stop_bit = Label(conf_window, text='Стоп бит:', font=("Calibri", 15))
    label_stop_bit.grid(row=5, column=0)
    bit_stop = Combobox(conf_window)
    bit_stop['values'] = STOPBITS
    bit_stop.current(0)
    bit_stop.grid(row=5, column=1)

    ##-- Настройки сохраняются
    def clicked():
        if validation(name, com_port, speed_b, size_b, parity_b, bit_stop, ser):
            conf_window.destroy()

    """Кнопка завершения настроек"""
    button = Button(conf_window, text="OK", command=clicked)
    # button.focus_set()
    # button.bind('<Button-1>', clicked)
    # button.bind('<Return>', clicked)
    button.grid(column=2)
    conf_window.mainloop()

```

validation.py:

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
from tkinter.messagebox import *
from my_package.configurations import BAUDRATES, BYTESIZES, PARITIES, STOPBITS

cut_port = []
##---Обрезаем полное имя COM-порта до <COM(цифра)>
def cut_port_name(str):
    global cut_port
    for i in range(len(str)):
        cut_port.append(str[i])
        cut_port[i] = cut_port[i].device
    return cut_port

def validation(name, com_port, speed_b, size_b, parity_b, bit_stop, ser):
    """
    Валидация параметров COM-порта
    """
    username = name.get()
    if not username:
        showerror("Username isn't define.", "Пожалуйста, введите имя")
        return False
    ser.username = username
    port = com_port.get()

```

```

if port not in cut_port:
    showerror("Bad COM-port.", port + " не существует")
    return False
ser.port = port
speed = speed_b.get()
# speed_u = unicode(speed, 'utf-8')
if int(speed) not in BAUDRATES:
    showerror("Bad baudrate.", speed + " не существует")
    return False
ser.baudrate = speed
byte_size = size_b.get()
# byte_size_u = unicode(byte_size, 'utf-8')
if int(byte_size) not in BYTESIZES:
    showerror("Bad bytesize.", byte_size + " не существует")
    return False
ser.bytesize = int(byte_size)
parity = parity_b.get()
if parity not in PARITIES:
    showerror("Bad parity.", parity + " не существует")
    return False
ser.parity = parity
stopbits = bit_stop.get()
# stopbits_u = unicode(stopbits, 'utf-8')
# if stopbits_u.isnumeric() == False or float(stopbits) not in STOPBITS:
try:
    if float(stopbits) not in STOPBITS:
        showerror("Bad stopbit.", stopbits + " не существует")
        return False
except:
    showerror("Bad stopbit.", stopbits + " не существует")
    return False
ser.stopbits = float(stopbits)
return True

```

chat.py:

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
import threading
import time
from datetime import datetime
from tkinter import *

def chat(ser):
    global out_flag
    global tr_in
    global in_list

    ##-- Разрешение на запуск команд проверки соединения
    global permission_check_connect
    permission_check_connect = 0

    # -- массив полученных строк
    in_list = []
    # -- признаки занятости ввода-вывода
    out_flag = []

    global result_available
    result_available = threading.Event()

    def give_username():
        while ser.another_username == None:

```

```

        time.sleep(2)
    if ser.is_open:
        # result_available.wait(timeout=3)
        ser.ft_write("Username" + str(ser.username))
    pass

    ##-- Буффер для команд
    global buffer_for_comand_message
    buffer_for_comand_message = []
    ## counter - счетчик(строка в listbox)
    ## -- Отправленные сообщения таким образом становятся синими
    global counter
    counter = 0
    def check_connect():
        global counter
        while True:
            if ser.is_open: # and permission_check_connect:
                time.sleep(10)
                try:
                    listbox_command.insert(END, "[" + datetime.strftime(datetime.now(),
"%H:%M:%S") + "]" + "ACK_LINKACTIVE")
                except:
                    pass
                    buffer_for_comand_message.append "[" + datetime.strftime(datetime.now(),
"%H:%M:%S") + "]" + "ACK_LINKACTIVE"
                    # listbox.insert(END, "[" + datetime.strftime(datetime.now(), "%H:%M:%S")
+ "]" + "ACK_LINKACTIVE")
                    # listbox.itemconfig(counter, {'fg': 'gray'})
                    ser.ft_write("ACK_LINKACTIVE")
                    # counter += 1
                    # time.sleep(10)

    global open_button_clicked
    ##-- UPLINK-кадр
    ##-- Кадр-запрос на разрешение соединения
    def try_connect():

        global result_available

        global counter
        global open_button_clicked
        open_button_clicked = 1
        while True:
            time.sleep(1)
            if ser.is_open and open_button_clicked:
                listbox.insert(END, "[" + datetime.strftime(datetime.now(), "%H:%M:%S") + "]"
+ "Запрос на соединение")
                # listbox.itemconfig(counter, {'fg': 'gray'})
                counter += 1
                try:
                    listbox_command.insert(END, "[" + datetime.strftime(datetime.now(),
"%H:%M:%S") + "]" + "UPLINK")
                except:
                    pass
                    buffer_for_comand_message.append "[" + datetime.strftime(datetime.now(),
"%H:%M:%S") + "]" + "UPLINK"
                    ser.ft_write("UPLINK")

                # result_available.set()

            open_button_clicked = 0

```

```

        # time.sleep(1)
    pass

##-- Функция:
##-- Если через 10 секунд после передачи UPLINK не пришел ответ
##-- то выводится сообщение о невозможности соединения
    global ACK_UPLINK_NOTCOME
    ACK_UPLINK_NOTCOME = 1
    def bad_connect():
        global counter
        counter_1 = 0
        while True:
            if ser.is_open:
                time.sleep(10)
                if ACK_UPLINK_NOTCOME == 1 and counter_1 == 0:
                    listbox.insert(END, "[" + datetime.strftime(datetime.now(), "%H:%M:%S")
+ "]" + "Соединение не установлено")
                    counter_1 += 1
                    counter += 1
                else:
                    counter_1 = 0
                    time.sleep(1)
            pass

    global in_st
    in_st = []
    # функция приема строки
    def fn_in():
        global counter
        global in_list
        global in_st

        global permission_check_connect
        global ACK_UPLINK_NOTCOME

        counter_temp = 0
        while 1:
            if ser.is_open:
                # --ждем прихода к нам строки
                while ser.in_waiting > 0:
                    if ser.is_open:
                        data_to_read = ser.in_waiting
                        in_st = ser.ft_read(data_to_read)
                        if in_st == "ACK_LINKACTIVE":
                            try:
                                listbox_command.insert(END, "[" +
datetime.strftime(datetime.now(), "%H:%M:%S") + "]" + ">>> LINKACTIVE")
                            except:
                                pass
                                buffer_for_comand_message.append("[" +
datetime.strftime(datetime.now(), "%H:%M:%S") + "]" + ">>> LINKACTIVE")
                                # listbox.insert(END, "[" + datetime.strftime(datetime.now(),
"%H:%M:%S") + "]" + "LINKACTIVE")
                                # listbox.itemconfig(counter, {'fg': 'gray'})
                                # counter += 1
                                in_st = []
                                elif in_st[:8] == "Username":
                                    ser.another_username = in_st[8:]
                                    in_st = []
                                elif in_st == "UPLINK":
                                    listbox.insert(END, "[" + datetime.strftime(datetime.now(),

```

```

"%H:%M:%S") + "]" + ">>> Соединение установлено")
    counter += 1
    # listbox.itemconfig(counter, {'fg': 'gray'})
    permission_check_connect = 1
    ACK_UPLINK_NOTCOME = 0
    try:
        listbox_command.insert(END, "[" +
datetime.strftime(datetime.now(), "%H:%M:%S") + "]" + ">>> ACK_UPLINK")
    except:
        pass
        buffer_for_comand_message.append "[" +
datetime.strftime(datetime.now(), "%H:%M:%S") + "]" + ">>> ACK_UPLINK")
        # if counter_temp == 0:
        #     ser.ft_write("UPLINK")
        #     counter_temp = 1
        in_st = []
    else:
        if in_st != '':
            in_list.append(in_st)
            time.sleep(1)    ##-- CPU не будет нагреваться до 100C

## -- запустить поток приема
global start_thread
start_thread = 0
tr_in = threading.Thread(target=fn_in)
tr_in.daemon = True

thread_2 = threading.Thread(target=check_connect)
thread_2.daemon = True

thread_3_name = threading.Thread(target=give_username)
thread_3_name.daemon = True

thread_4 = threading.Thread(target=try_connect)
thread_4.daemon = True

thread_5_bad_connect = threading.Thread(target=bad_connect)
thread_5_bad_connect.daemon = True

## -- запустить основной поток
def fn_out():
    global out_flag
    out_flag = 1

##--Отправление сообщений через кнопку "Отправить"
global buffer_for_source_message
buffer_for_source_message = []

def fn_send():
    global counter
    # global user_name
    out_st = enter.get()
    if len(out_st) > 0:
        ser.ft_write((out_st + '\r\n'))
        listbox.insert(END, "[" + datetime.strftime(datetime.now(), "%H:%M:%S")+ "]" "
+ ser.username + ": " + out_st)
        listbox.itemconfig(counter, {'fg': 'blue'})
        counter += 1
        buffer_for_source_message.append "[" + datetime.strftime(datetime.now(),
"%H:%M:%S")+ "]" " + ser.username + ": " + out_st)
    try:

```

```

        listbox_source.insert(END, "[" + datetime.strftime(datetime.now(),
"%H:%M:%S")+ "]" + ser.username + ": " + out_st)
    except:
        pass
    enter.delete(0, END)

## == вывести строки в листбокс
global buffer_for_dest_message
buffer_for_dest_message = []

def fn_disp():
    global counter
    global out_flag
    while len(in_list) > 0:
        st = in_list.pop(0)
        if ser.another_username != None:
            listbox.insert(END, "[" + datetime.strftime(datetime.now(), "%H:%M:%S")+ "]"
+ ser.another_username + ": " + st)
            listbox.itemconfig(counter, {'fg': 'red'})
            counter += 1
            buffer_for_dest_message.append "[" + datetime.strftime(datetime.now(),
"%H:%M:%S")+ "]" + ser.another_username + ": " + st)
        else:
            listbox.insert(END, "[" + datetime.strftime(datetime.now(), "%H:%M:%S")+ "]"
+ ">>> " + st)
            listbox.itemconfig(counter, {'fg': 'red'})
            counter += 1
            buffer_for_dest_message.append "[" + datetime.strftime(datetime.now(),
"%H:%M:%S")+ "]" + ">>> " + st)
        try:
            listbox_dest.insert(END, st)
        except:
            pass
    if out_flag:
        fn_send()
        out_flag = 0
    window.after(100, fn_disp)

window = Tk()
window.geometry('716x400')

scrollbar = Scrollbar(window)
scrollbar.pack(side=RIGHT, fill=Y)

listbox = Listbox(window, yscrollcommand=scrollbar.set, font=('Calibri', 12))
listbox.place(x=0, y=0, width=600, height=340)

scrollbar.config(command=listbox.yview)

enter = Entry(window, font=('Calibri', 15))
enter.place(x=0, y=340, width=600, height=40)

def open_port():
    global counter
    global tr_in
    global start_thread
    state = DISABLED

    global open_button_clicked

```



```

if ser.is_open == False:
    ser.open()

    open_button_clicked = 1

    if ser.is_open:
        listbox.insert(END, "Port " + ser.port + " is opened")
        button_open.config(text="Заккрыть порт")
        button_display.config(state=NORMAL)
        counter += 1
        if start_thread == 0:
            tr_in.start()
            thread_2.start()
            thread_3_name.start()
            thread_4.start()
            thread_5_bad_connect.start()
            start_thread = 1
    else:
        ser.close()
        if ser.is_open == False:
            listbox.insert(END, "Port " + ser.port + " is closed")
            button_open.config(text="Открыть порт")
            button_display.config(state=DISABLED)
            counter += 1
button_open = Button(window, text="Открыть порт", command=open_port)
button_open.focus_set()
button_open.place(x=600,y=0, width=100, height=40)

global counter_info_window
counter_info_window = 0
def about_program():
    """Меню-справка о создателях программы
    Количество открытых окон не должно превышать одного"""
    global counter_info_window
    if counter_info_window == 0:
        temp_window = Toplevel(window)
        def close_window():
            global counter_info_window
            counter_info_window -= 1
            temp_window.destroy()
        temp_window.protocol("WM_DELETE_WINDOW", close_window)
        temp_window.title('О программе')
        temp_window.geometry('300x100')
        student_1 = Label(temp_window, text="Анастасия Молева", font=('Arial', 15))
        student_1.grid(row=0,column=0)
        student_2 = Label(temp_window, text="Матиенко Андрей", font=('Arial', 15))
        student_2.grid(row=1,column=0)
        student_3 = Label(temp_window, text="Белоусов Евгений", font=('Arial', 15))
        student_3.grid(row=2,column=0)
        counter_info_window += 1

mainmenu = Menu(window)
window.config(menu=mainmenu)
mainmenu.add_command(label="О программе", command=about_program)

##--Исходящие сообщения(source_message)
global counter_source_window
counter_source_window = 0
def source_message():
    """Окно - Отправленные сообщения
    Если окно открыто, то кнопка становится недоступной"""
    global listbox_source

```

```

global counter_source_window
if counter_source_window == 0:
    window_source_message = Toplevel(window)
    def close_window():
        global counter_source_window
        counter_source_window -= 1
        window_source_message.destroy()
        button_source_message.config(state='normal')
    window_source_message.protocol("WM_DELETE_WINDOW", close_window)
    window_source_message.title('Исходящие сообщения')
    window_source_message.geometry('600x400+500+200')
    listbox_source = Listbox(window_source_message, font=('Calibri', 12))
    listbox_source.place(x=0, y=0, width=600, height=340)
    counter_source_window += 1
    button_source_message.config(state=DISABLED)
    for i in buffer_for_source_message:
        listbox_source.insert(END, i)

button_source_message = Button(window, text='Исходящие', command=source_message,
state='normal')
button_source_message.place(x=600,y=200, width=100,height=40)
##-----

##--Приходящие сообщения(destination_message)
global count_dest_window
count_dest_window = 0
def dest_message():
    """Окно - Пришедшие сообщения
    Если окно открыто, то кнопка становится недоступной"""
    global listbox_dest
    global count_dest_window
    if count_dest_window == 0:
        window_dest_message = Toplevel(window)
        def close_window():
            global count_dest_window
            count_dest_window -= 1
            window_dest_message.destroy()
            button_dest_message.config(state='normal')
        window_dest_message.protocol("WM_DELETE_WINDOW", close_window)
        window_dest_message.title('Приходящие сообщения')
        window_dest_message.geometry('600x400+800+200')
        listbox_dest = Listbox(window_dest_message, font=('Calibri', 12))
        listbox_dest.place(x=0, y=0, width=600, height=340)
        button_dest_message.config(state=DISABLED)
        for i in buffer_for_dest_message:
            listbox_dest.insert(END, i)
        count_dest_window += 1

button_dest_message = Button(window, text='Приходящие', command=dest_message,
state='normal')
button_dest_message.place(x=600,y=250,width=100,height=40)

##-- Окно команд(LINKACTIVE...)

global count_command_window
count_command_window = 0

def command_button():
    """Окно - сообщения команд
    Если окно открыто, то кнопка становится недоступной"""
    global listbox_command

```

```

global count_command_window
if count_command_window == 0:
    window_command_message = Toplevel(window)

    def close_window():
        global count_command_window
        count_command_window -= 1
        window_command_message.destroy()
        button_command_message.config(state='normal')

    window_command_message.protocol("WM_DELETE_WINDOW", close_window)
    window_command_message.title('Команды')
    window_command_message.geometry('600x400+800+200')
    listbox_command = Listbox(window_command_message, font=('Calibri', 12))
    listbox_command.place(x=0, y=0, width=600, height=340)
    button_command_message.config(state=DISABLED)
    for i in buffer_for_comand_message:
        listbox_command.insert(END, i)
    count_command_window += 1

    button_command_message = Button(window, text='Команды', command=command_button,
state='normal')
    button_command_message.place(x=600, y=150, width=100, height=40)
    ###-----

    button_display = Button(window, text='Отправить', command=fn_out, state=DISABLED,)
    button_display.place(x=600, y=340, width=100, height=40)
    window.after(10, fn_disp)
    window.mainloop()

```