Московский государственный технический университет им. Н.Э. Баумана

|  |  |
| --- | --- |
| Утверждаю: |  |
| Галкин В.А. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г. |

**Курсовая работа по дисциплине**

**«Cетевые 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. | \_\_\_\_\_\_\_\_\_\_\_ |

Москва - 2020 г.

# **Общие сведения**

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

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

1. **Назначение разработки**

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

# **Описание логической структуры**

## **Алгоритм интерфейсной части прораммы**

Алгоритм интерфейсной части приведен на рисунке.



## **Алгоритм передачи сообщения**



## **Алгоритм приема сообщения**



# **Используемые технические средства**

Программа должна работать на IBM-совместимой ЭВМ следующей конфигурации:

1. Центральный процессор Pentium 4 или выше;
2. Объем оперативной памяти 512 Мб;
3. Видеоадаптер и монитор VGA и выше;
4. Стандартная клавиатура;
5. Свободного пространства на жестком диске 2Мб;

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

# **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.

# **Спецификация данных**

## **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 |

# **Спецификация функций**

**def** main(): - головная функция программы

**def** configure\_window(ser): - окно настроек

**def** clicked(): - нажатие на кнопку «ОК»

**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*

## **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

## **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 буффер

**def** in\_waiting(self): - возвращает количество байт в input буффере

**def** ft\_read(self, size=1): - чтение из буффера

1. **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** \_\_buildin\_\_  
 **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: {!r}"**.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 if not...* **except** TypeError:  
 **raise** ValueError(**"Not a valid timeout: {!r}"**.format(timeout))  
  
 self.\_write\_timeout = timeout  
 **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-port"""** 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(строчка 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=**"Закрыть 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()