Московский государственный технический университет им. Н.Э. Баумана

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

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

Описание программы
(вид документа)

писчая бумага
(вид носителя)

26
(количество листов)

ИСПОЛНИТЕЛИ:

студенты группы ИУ5-61Б
Болгова А.В. _____
Фонканц Р.В. _____
Попов И.А. _____

Москва - 2020 г.

Оглавление

1. Общие сведения

Наименование: «Программа для обмена файлами и текстовыми сообщениями».

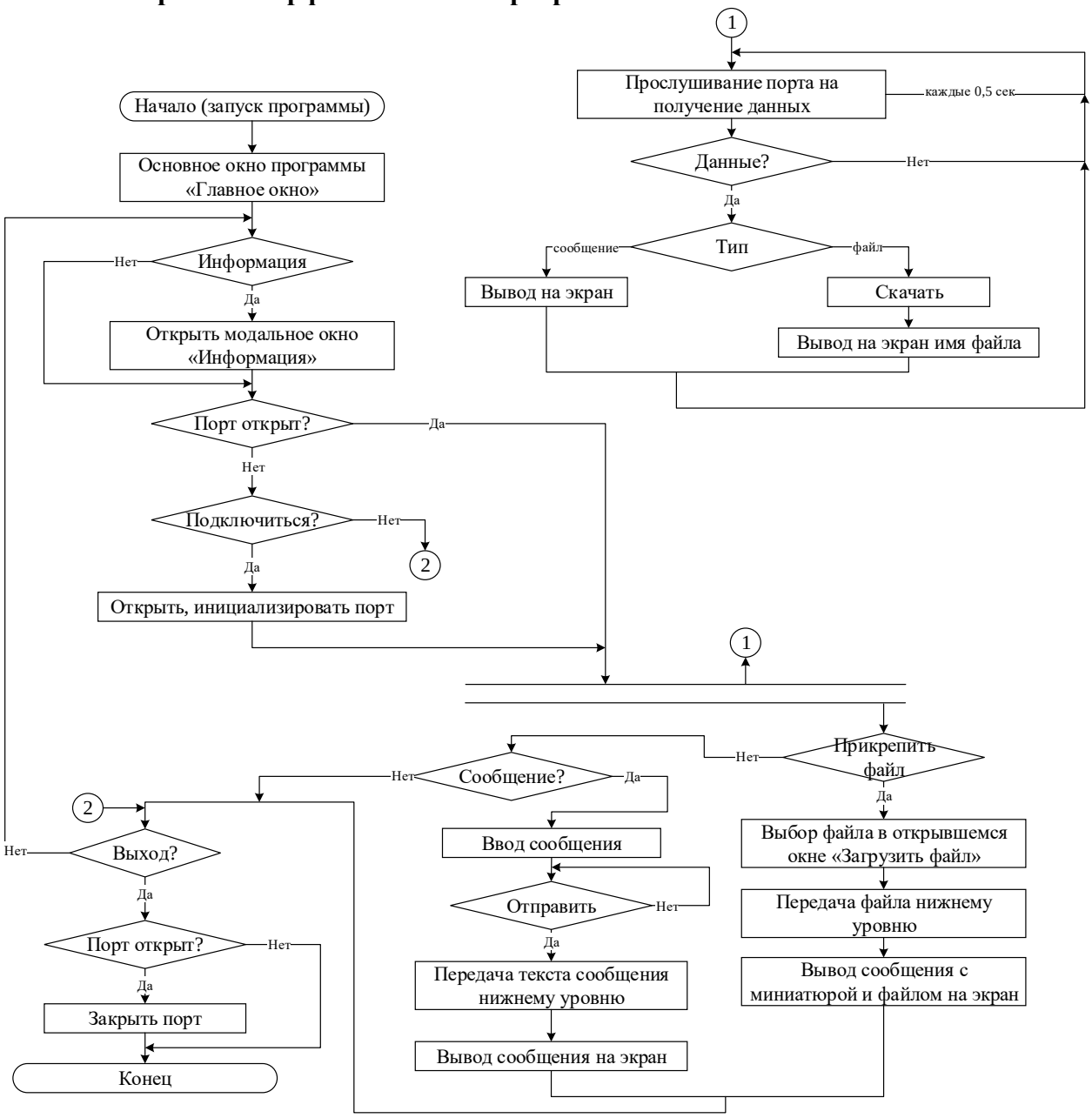
Программа выполняется на языке программирования Python и работает под управлением операционной системы Windows 7 и выше/Linux.

2. Назначение разработки

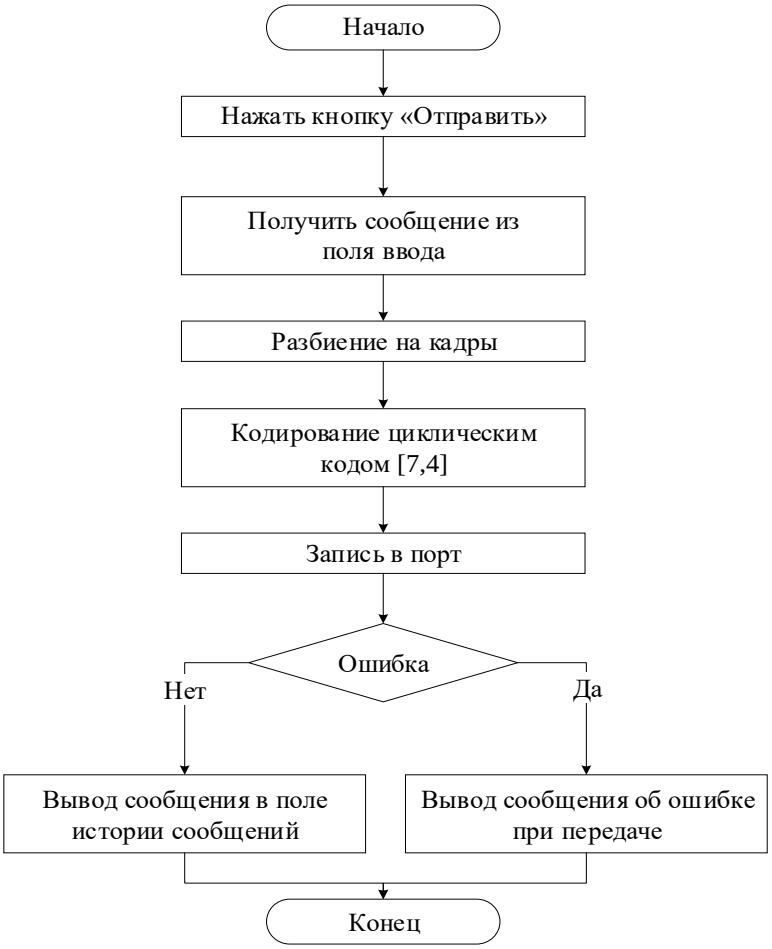
Программа должна реализовывать функцию передачи текстовых сообщений и файлов между двумя ПЭВМ, соединенными через интерфейс RS-232C с использованием нуль-модемного кабеля.

3. Описание логической структуры

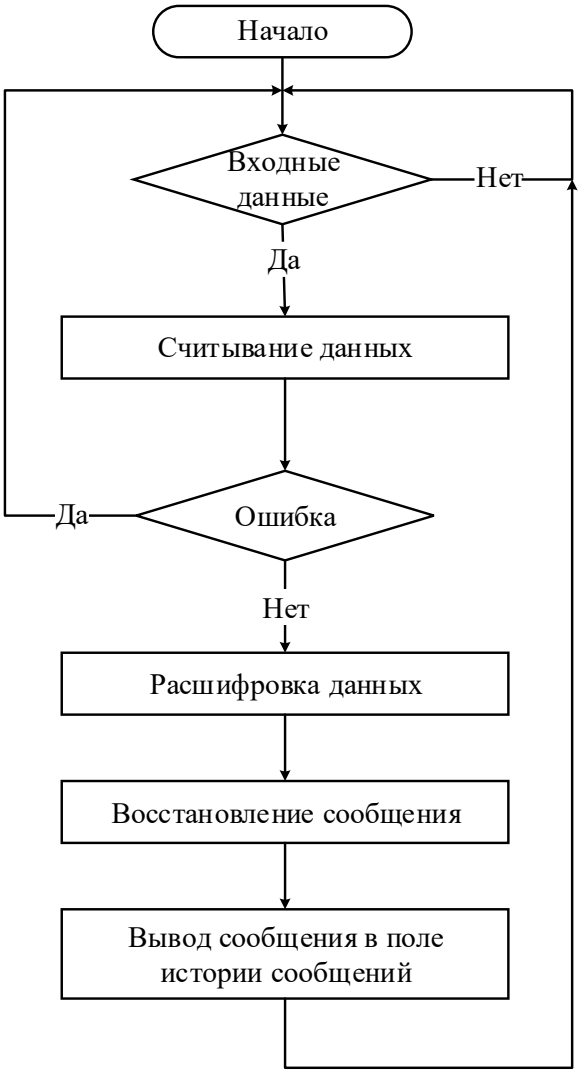
3.1. Алгоритм интерфейсной части программы



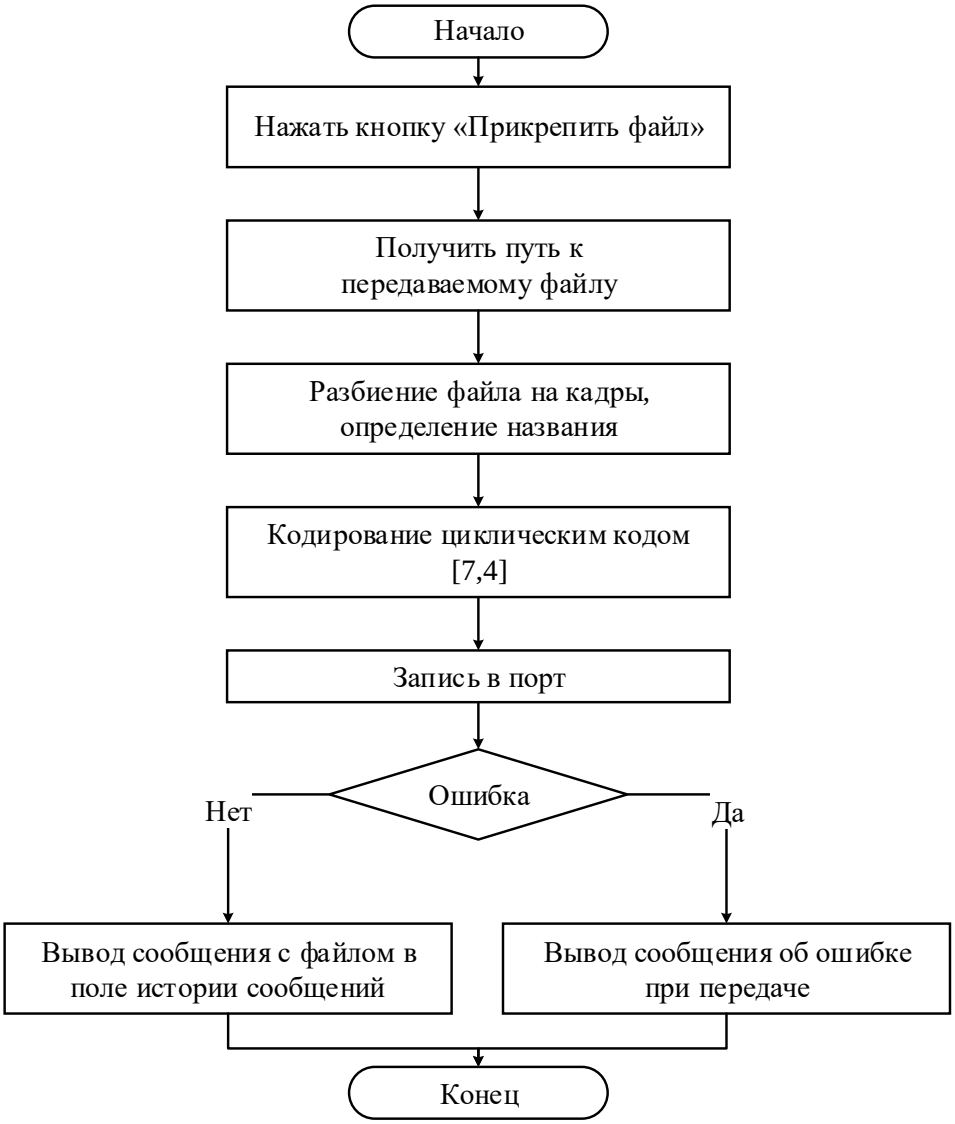
3.2. Алгоритм передачи сообщения



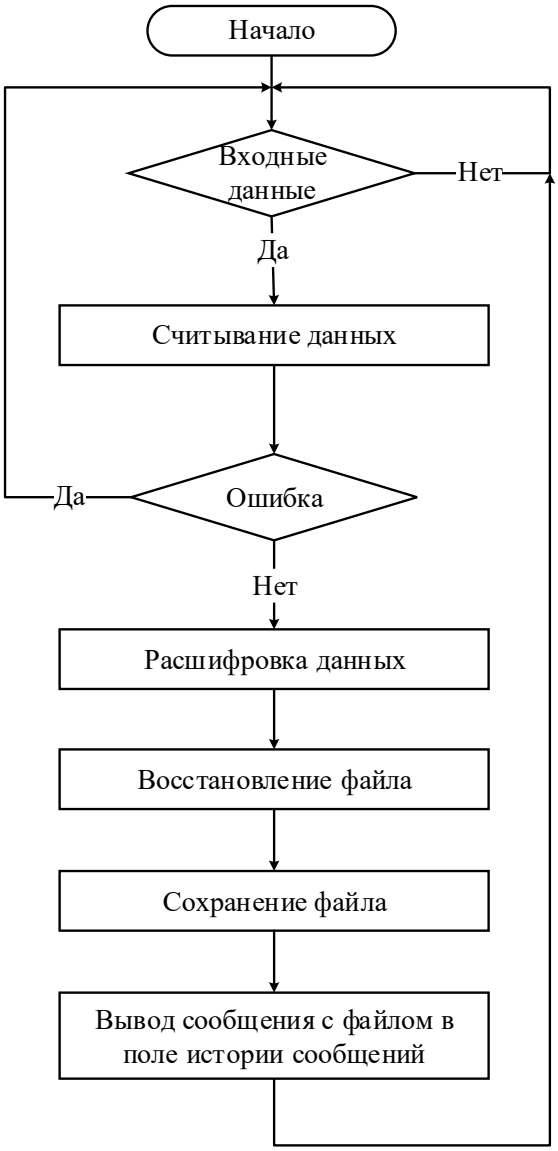
3.3. Алгоритм приема сообщения



3.4. Алгоритм передачи файла



3.5. Алгоритм приема файла



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 являются:
- текстовое сообщение, набранное пользователем с клавиатуры
- файл, выбранный пользователем

5.2. Выходные данные.

Выходными данными являются:
- текст переданного сообщения на ПЭВМ

- файл в каталоге принимающей ПЭВМ
- сообщения об ошибках и выполнении передачи

6. Спецификация данных

6.1. Внутренние данные

Заголовочный кадр:

2 бита	6 бит	24 бита	1 бит	7 бит	128 б
Флаг типа кадра	Идентификатор сообщения	Количество кадров в сообщении	Тип сообщения	Количество значащих символов в кадре	данные

Информационный кадр:

2 бита	6 бит	24 бита	1 бит	7 бит	128 б
Флаг типа кадра	Идентификатор сообщения	Номер текущего кадра	Тип сообщения	Количество значащих символов в кадре	данные

Служебные кадры:

2 бита	6 бит	132 байта
Флаг типа кадра	Тип служебного кадра	Заполнитель

`def get_port_parameters()`	*Функция парсит config.json и возвращает значения параметров открываемых COM-портов*
`def portinit(com)`	*Функция инициализация порта*
`def ser_open()`	*Функция открытия порта*
`def ser_close()`	*Функция закрытия порта*
`def ser_write(data)`	*Функция записи данных в порт*
`def ser_read(size)`	*Функция чтения данных из порта*
`def send(text, file=False, file_name=None)`	*Функция подготовки данных для отправки (разбиение на кадры и шифрование циклическим кодом)*
`def receive(frames_encoded)`	*Функция дешифровки принятых кадров и восстановления данных*
`def send_file(file)`	*Функция подготовки данных файла и его названия для отправки*
`def recieve_file(file)`	*Функция восстановления и сохранения файла*
`def listener_connection(state)`	*Функция передачи состояния порта клиенту*
`def listener_user_connection(state)`	*Функция передачи состояния соединения с собеседником клиенту*
`def listener_transmission_failed()`	*Функция оповещения клиента об ошибке при передаче сообщения*

6.2. Функции класса InfoDialog

`def __init__(self, parent=None)`	*Функция инициализации окна*
`self.setupUi(self)`	*Функция инициализации графических элементов интерфейса окна*
`def init_handlers(self)`	*Привязка обработчика события нажатия кнопки «ОК»*
`self.setFixedSize(self.size())`	*Запрет изменения размеров окна*

6.3. Функции класса PortListener

`def line_recieved(self, text)`	*Функция-триггер на получение портом текстового сообщения. Посылает сигнал в главное окно функции вывода сообщения*
`def file_recieved(self, text)`	*Функция-триггер на получение портов файла. Посылает сигнал в главное окно функции вывода сообщения о файле*
`def connecting(self, state)`	*Функция-триггер на физическое соединение с собеседником*
`def user_connecting(self, state)`	*Функция-триггер на подключение к COM-порту*
`def transmission_failed(self)`	*Функция-триггер на ошибку при отправке сообщения или файла*

6.4. Функции класса ChatApp

`def __init__(self)`	*Функция инициализации окна*
`self.setupUi(self)`	*Инициализирует графические элементы интерфейса*
`self.init_handlers()`	*Связывает кнопки с обработчиками, какие функции вызываются при нажатии*
`self.init_toolbar()`	*Связывает элементы тулбара (меню) с обработчиками нажатия на (подключение, выход, информация)*
`def closeEvent(self, event: QtGui.QCloseEvent)`	*Переопределение события close (X)*
`def close_connection(self)`	*Закрытие соденинния и портов*
`def init_connection(self)`	*Открытие портов, установка соединения, запуск прослушки портов*
`def create_dialog(self)`	*Создание окна Информации*
`def send_message(self)`	*Отправка сообщения нижнему уровню и его вывод на экран, очистка поля ввода/редактирования сообщения*
`def open_dialog(self)`	*Открытие модального диалогового окна выбора файла для отправки, отправка файла*
`def save_dialog(self)`	*Открытие модального диалогового окна выбора директории для сохранения скачиваемого файла*
`def show_service(self, content)`	*Добавление служебного сообщения (об ошибке) в основное окно*
`@QtCore.pyqtSlot(str)` `def show_message(self, content)`	*Добавление нового сообщения в основное окно (в т.ч. при получении сообщения)*
`@QtCore.pyqtSlot(str)` `def show_file(self, content)`	*Добавление строки файла в основное окно (в т.ч. при получении файла)*
`def listen(self)`	*Создает объект port_listener в главном окне (связующее звено с канальным уров нем), и связывает сигналы [line - получено сообщение, file - получен файл] с show_message и show_file*
`def show_disconnect(self, state)`	*В случае разрыва соединения изменяет статус на "Соединение потеряно. Нажмите подключиться" и блокирует кнопки Отправить и Прикрепить файл*
`def block_buttons(self)`	*Блокирует кнопки Отправить и Прикрепить файл*
`def unblock_buttons(self)`	*Разблокирует кнопки Отправить и Прикрепить файл*
`def show_user_connect(self, state)`	*Меняет состояние иконки подключения собеседника*

6.5. Функции в классе SerialBase

Инициализация:	
def port(self, port)	*Задаёт наименование порта*
def baudrate(self, baudrate)	*Задаёт пропускную способность в бодах*
def bytesize(self, bytesize)	*Задаёт размер байта*
def parity(self, parity)	*Задаёт бит четности*
def stopbits(self, stopbits)	*Задаёт стопбит*
def timeout(self, timeout)	*Задаёт таймаут чтения*
def writeTimeout(self, timeout)	*Задаёт таймаут записи*
Настройки:	
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 __repr__(self)	*Отобразить всю информацию о порте*

6.6. Функции в классе Serial

Инициализация:	
def open(self)	*Функция открытия порта*
def _reconfigure_port(self)	*Set communication parameters on opened port (Функция настроить порт)*
def close(self)	*Функция закрытия порта*
def _cancel_overlapped_io(self, overlapped)	*Функция прекращения чтения/записи данных*
def cancel_read(self)	*Функция, которая ссылается на _cancel_overlapped_io при чтении*
def cancel_write(self)	*Функция, которая ссылается на _cancel_overlapped_io при записи*
def write(self, data)	*Функция записи в буффер*
def in_waiting(self)	*Функция, которая возвращает количество байт в input буффере*
def read(self, size=1)	*Функция чтения из буффера*

7. Листинг основных функций

client.py:

```
import sys
import os
import datetime
from PyQt5 import QtGui, QtCore, QtWidgets

from gui import window
from utils import info_dialog

import phizical
import channel


# ----------------------- ChatApp ----------------------------
# -- класс главного окна | gui/window.py, gui/ui/window.ui --
# ------------------------------------------------------------
# Переменные:
# infoDialog        модальное диалоговое окно информации
# bSend             кнопка Отправить (сообщение)
# bSendFile         кнопка Выбрать файл (и отправить)
# textList          основное окно с сообщениями и файлами
# mExit             кнопка тулбара Выход
# mConnect          кнопка тулбара Подключиться
# mInfo             кнопка тулбара Информация
# message           окно набора текстового сообщения
```

```
# port_listener     определен в channel.py - сигнализирует
#                   о пришедшем сообщении/файле в гл. окно
# ------------------------------------------------------------


class ChatApp(QtWidgets.QMainWindow, window.Ui_MainWindow):
    # setupUI           инициализирует графические элементы интерфейса
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.init_handlers()
        self.init_toolbar()
        self.block_buttons()
        self.infoDialog = None
        self.userStatus = False
        self.statusBar.showMessage("Чтобы установить соединение и начать работу,
нажмите Подключиться")

    # init_handlers     связывает кнопки с обработчиками, какие функции вызываются при
нажатии
    def init_handlers(self):
        self.bSend.clicked.connect(self.send_message)
        self.bSend.setAutoDefault(True)
        self.message.returnPressed.connect(self.bSend.click)
        self.bSendFile.clicked.connect(self.open_dialog)
        self.textList.itemDoubleClicked.connect(self.save_dialog)

    # init_toolbar      связывает элементы тулбара с обработчиками (подключение, разрыв
связи, выход, информация)
    def init_toolbar(self):
        self.mExit.triggered.connect(self.close_connection)
        self.mConnect.triggered.connect(self.init_connection)
        self.mDisonnect.triggered.connect(phizical.ser_close)
        self.mInfo.triggered.connect(self.create_dialog)

    # closeEvent        переопределение события close (X)
    def closeEvent(self, event: QtGui.QCloseEvent):
        self.close_connection()
        event.accept()

    # close_connection  закрытие соденинения и портов
    def close_connection(self):
        self.statusBar.showMessage("Отключение...")
        phizical.ser_close()
        print("Connection closed")
        QtWidgets.qApp.quit()

    # init_connection   открытие портов, установка соединения, запуск прослушки портов
    def init_connection(self):
        try:
            phizical.portinit()
            phizical.ser_open()
            phizical.ser_read()
            print("Connection open! Start reading")
            self.show_service("Подключено")
            self.statusBar.showMessage("Порт подключен")
            self.unblock_buttons()
            self.listen()
        except:
            self.show_service("Невозможно установить соединение")

    # create_dialog     показ окна Информации
```

```
    def create_dialog(self):
        if not self.infoDialog:
            self.infoDialog = info_dialog.InfoDialog()
        self.infoDialog.show()

    # send_message      отправка сообщения, очистка окна
    def send_message(self):
        try:
            message = self.message.text()
            if len(message) > 0:
                content = f"{datetime.datetime.now().strftime('%H:%M')} <Вы>:
{message}"

                phizical.ser_write(channel.send(message))
                self.show_message(content)

                self.message.clear()
            else:
                self.show_service('Нельзя отправить пустое сообщение')
        except:
            self.show_service('Невозможно отправить сообщение')

    # open_dialog       открытие модального диалогового окна выбора файла для отправки,
отправка файла
    def open_dialog(self):
        filepath = QtWidgets.QFileDialog.getOpenFileName(self, "Загрузить файл", "")[0]
        if filepath:
            try:
                f = QtCore.QFileInfo(filepath)
                print(f.absolutePath())

                phizical.ser_write(channel.send_file(f.absoluteFilePath()))
                self.show_file(f.fileName())
            except:
                self.show_service('Невозможно прикрепить файл')

    # save_dialog       открытие модального диалогового окна выбора директории для
сохранения скачиваемого файла
    def save_dialog(self):
        item = self.textList.currentItem()
        file_name = item.text()

        if item.data(QtCore.Qt.UserRole) == 1:
            directory = QtWidgets.QFileDialog.getExistingDirectory(self, "Выбрать папку
для сохранения", "")
            if directory:
                new_dir = directory + '/' + file_name
                if not os.path.isfile(new_dir):
                    os.replace("../downloads/" + file_name, new_dir)
                else:
                    i = 1
                    new_file_name = file_name[:file_name.rfind('.')] + \
                                    '(' + str(i) + ')' +
file_name[file_name.rfind('.'):]
                    new_dir = directory + '/' + new_file_name
                    while os.path.isfile(new_dir):
                        new_file_name = file_name[:file_name.rfind('.')] + \
                                        '(' + str(i) + ')' +
file_name[file_name.rfind('.'):]
                        new_dir = directory + '/' + new_file_name
                        i = i + 1
                    else:
```

```
                    os.replace("../downloads/" + file_name, new_dir)

                print(directory, file_name)

    # show_service      добавление служебного сообщения (об ошибке) в основное окно
    @QtCore.pyqtSlot(str)
    def show_service(self, content):
        item = QtWidgets.QListWidgetItem()
        item.setText(content)
        item.setForeground(QtGui.QColor(27, 151, 243))
        self.textList.addItem(item)
        self.textList.scrollToBottom()

    # show_message      добавление нового сообщения в основное окно
    @QtCore.pyqtSlot(str)
    def show_message(self, content):
        item = QtWidgets.QListWidgetItem()
        item.setText(content)
        self.textList.addItem(item)
        self.textList.scrollToBottom()

    # show_file         добавление строки файла в основное окно
    @QtCore.pyqtSlot(str)
    def show_file(self, content):
        iconfile = QtGui.QIcon('../gui/icon/download.png')

        item = QtWidgets.QListWidgetItem()
        item.setIcon(iconfile)
        item.setText(content)
        item.setData(QtCore.Qt.UserRole, 1)

        self.textList.addItem(item)
        self.textList.setIconSize(QtCore.QSize(32, 32))
        self.textList.scrollToBottom()

    # show_disconnect   в случае разрыва соединения изменяет статус на "Соединение
потеряно. Нажмите
    #                   подключиться" и блокирует кнопки Отправить и Прикрепить файл
    @QtCore.pyqtSlot(bool)
    def show_disconnect(self, state):
        self.statusBar.showMessage("Соединение потеряно. Порт не найден")
        self.block_buttons()

    # block_buttons     делает кнопки Отправить и Прикрепить файл некликабельными
    def block_buttons(self):
        self.bSend.setEnabled(False)
        self.bSendFile.setEnabled(False)

    # unblock_buttons   делает доступными для нажатия кнопки Отправить и Прикрепить
файл
    def unblock_buttons(self):
        self.bSend.setEnabled(True)
        self.bSendFile.setEnabled(True)

    # show_user_connect меняет состояние подключения собеседника в тулбаре
    @QtCore.pyqtSlot(bool)
    def show_user_connect(self, state):
        if state != self.userStatus and state:
            self.userStatus = True
            self.pixmap = QtGui.QPixmap("../gui/icon/online.png")
            self.statusLabel.setPixmap(self.pixmap)
```

```
        if state != self.userStatus and not state:
            self.userStatus = False
            self.pixmap = QtGui.QPixmap("../gui/icon/offline.png")
            self.statusLabel.setPixmap(self.pixmap)

    # listen            создает объект port_listener в главном окне (связующее звено с
канальным уров-
    #                   нем), и связывает сигналы [line - получено сообщение, file -
получен файл] с
    #                   [line - получено сообщение | show_message]
    #                   [file - получен файл | show_file]
    #                   [connected - состояние соединения, реагирует когда соединения
нет | show_disconnect]
    #                   [user_connected - состояние подключения собеседника |
show_user_connect]
    #                   [transmission_error - ошибка передачи файла или сообщения |
show_service]
    def listen(self):
        self.port_listener = channel.plistener
        self.port_listener.line.connect(self.show_message)
        self.port_listener.file.connect(self.show_file)
        self.port_listener.connected.connect(self.show_disconnect)
        self.port_listener.user_connected.connect(self.show_user_connect)
        self.port_listener.transmission_error.connect(self.show_service)


if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)  # Новый экземпляр QApplication
    window = ChatApp()                      # Создает объект
    window.show()                           # Показывает окно
    sys.exit(app.exec_())                   # и запускает приложение
```

phizical.py:

```
# !/usr/bin/python


#import serial.tools.list_ports
import datetime
import threading
import time
import serial
import channel
# import ft_serial_1
import utils.parser
#from serial.tools import list_ports

# initialization and open the port

# possible timeout values:
#    1. None: wait forever, block call
#    2. 0: non-blocking mode, return immediately
#    3. x, x is bigger than 0, float allowed, timeout block call


"""ser_1 = serial.Serial()
# ser.port = "/dev/ttyUSB0"
ser_1.port = "COM6"
# ser.port = "/dev/ttyS2"
```

```
ser_1.baudrate = 115200
ser_1.bytesize = serial.EIGHTBITS  # number of bits per bytes
ser_1.parity = serial.PARITY_NONE  # set parity check: no parity
ser_1.stopbits = serial.STOPBITS_ONE  # number of stop bits
# ser.timeout = None  #block read
ser_1.timeout = 1  # non-block read
# ser.timeout = 2  #timeout block read
# ser_1.xonxoff = False  # disable software flow control
# ser_1.rtscts = False  # disable hardware (RTS/CTS) flow control
# ser_1.dsrdtr = False  # disable hardware (DSR/DTR) flow control
ser_1.writeTimeout = None  # timeout for write"""

ser_1 = serial.Serial()


def portinit(com=utils.parser.get_port_parameters()):
    # ser.port = "/dev/ttyUSB0"
    # ser.port = "/dev/ttyS2"
    ser_1.baudrate = com[2]
    ser_1.bytesize = serial.EIGHTBITS  # number of bits per bytes
    ser_1.parity = serial.PARITY_NONE  # set parity check: no parity
    ser_1.stopbits = serial.STOPBITS_ONE  # number of stop bits
    # ser.timeout = None          #block read
    ser_1.timeout = com[3]  # non-block read
    # ser.timeout = 2              #timeout block read
    # ser_2.xonxoff = False  # disable software flow control
    # ser_2.rtscts = False  # disable hardware (RTS/CTS) flow control
    # ser_2.dsrdtr = False  # disable hardware (DSR/DTR) flow control
    ser_1.writeTimeout = com[4]  # timeout for write
    try:
        ser_1.port = com[0]
        ser_1.open()
        ser_1.close()
    except:
        ser_1.port = com[1]


def ser_close():
    try:
        ser_1.close()
        channel.listener_connection(ser_1.isOpen())
    except Exception as e:
        print("error close serial port: " + str(e))
        exit()

priem = 0
read_delay = 0.5
frames = []


def ser_open():
    try:
        ser_1.open()
        channel.listener_connection(ser_1.isOpen())
    except Exception as e:
        print("error open serial port: " + str(e))
        exit()


def ser_write(binary_message):
    ser_1.flushInput()  # flush input buffer, discarding all its contents
    ser_1.flushOutput()  # flush output buffer, aborting current output
```

```
    channel.listener_connection(ser_1.isOpen())
    channel.listener_user_connection(ser_1.dsr)
    print('ser_write top', ser_1.isOpen(), ser_1.dsr)
    if ser_1.isOpen() and ser_1.dsr and ser_1.cts:
        for frame in binary_message:
            try:
                ser_1.write(frame)
                #print('WRITE', i, datetime.datetime.now())
                ser_1.flush()
            except Exception as e1:
                if not ser_1.cts:
                    channel.listener_transmission_failed()
                print("error communicating write...: " + str(e1))
    elif ser_1.isOpen() and not ser_1.cts:
        channel.listener_transmission_failed()
    else:
        channel.listener_connection(ser_1.isOpen())
        channel.listener_user_connection(ser_1.dsr)
        print("cannot open serial port, write", ser_1.isOpen(), ser_1.dsr)


def ser_read(ser=ser_1):
    global priem
    global frames
    global read_delay
    if ser.isOpen():
        if priem == 0:
            channel.listener_user_connection(ser.dsr)
            channel.listener_connection(ser_1.isOpen())
            print('ser_read top', ser_1.isOpen(), ser_1.dsr)
            #print(ser.cts, ser.dsr, ser.ri, ser.cd)
        if ser.in_waiting == 0 and priem == 1:
            priem = 0
            read_delay = 0.5
            channel.receive(frames)
            frames = []
        while ser.in_waiting > 0:
            if priem == 0:
                priem = 1
                read_delay = 0.0001
            print('in_waiting', ser_1.in_waiting)
            while ser.in_waiting >= 238:
                response = ser.read(238)
                #print('READ', ser_2.in_waiting, datetime.datetime.now())
                frames.append(response)
    else:
        channel.listener_connection(ser_1.isOpen())
        channel.listener_user_connection(False)
        print("cannot open serial port, read", ser_1.isOpen())
        return 1
    threading.Timer(read_delay, ser_read).start()
```

channel.py:

```
import random
import datetime

import utils.listener
plistener = utils.listener.PortListener()

def send(text, file=False, file_name=None):
```

```
# ------------ splitting on frames
    #print('framing...', datetime.datetime.now())
    if not file:
        text = text.encode('utf-8')
    frame_amount = len(text) // 128
    last_frame_len = len(text) % 128 - 1
    data_len = 127
    if frame_amount == 0:
        data_len = last_frame_len
    if last_frame_len < 0:
        frame_amount = frame_amount - 1
        last_frame_len = 127
    message_id = random.randint(1, 63)
    #print(hex(message_id+192))
    if file:
        frame_amount = frame_amount + 1
        data_len = 255
        file_name = bytes(file_name, encoding='utf-8')
        prim_frame = bytes([message_id + 192, frame_amount // 65536, frame_amount %
65536 // 256, frame_amount % 256, data_len]) + file_name
        prim_frame = prim_frame + bytes([ord('?') for i in range(128-len(file_name))])
        data_len = 127
        text = bytes([0 for i in range(128)]) + text
    else:
        prim_frame = bytes([message_id + 192, frame_amount // 65536, frame_amount %
65536 // 256, frame_amount % 256, data_len]) + text[:128]


    frames = []
    frames.append(prim_frame)
    for i in range(frame_amount):
        text = text[128:]
        if i == frame_amount-1:
            data_len = last_frame_len
        frames.append(bytes([message_id, i // 65536, i % 65536 // 256, i % 256,
data_len]) + text[:128])
    frames[len(frames)-1] = frames[len(frames)-1] + bytes([ord(')') for i in range(133-
len(frames[len(frames)-1]))])

# ------------ encoding [4,7]
    #print('encoding...', datetime.datetime.now())
    frames_encoded = []
    for frame in frames:
        encoding1 = frame
        #print('encoding1', encoding1, len(encoding1))
        frame_encoded = bytes([])                              # encoding1 - hole frame
[1 и 2 в названия х переменных - уровни фрагментации]
        while len(encoding1) > 0:
            if len(encoding1) > 3:
                encoding2 = int.from_bytes(encoding1[len(encoding1)-4:],
byteorder='big')   # encoding2 - 4 bytes
                encoding1 = encoding1[:len(encoding1)-4]
            else:
                encoding2 = int.from_bytes(encoding1, byteorder='big')  # encoding2 - 4
bytes
                encoding1 = ''
            mask = 15
            #print('encoding1', encoding1, len(encoding1))
            encoded2 = 0
            for i in range(8):                                          # encoding
each 4 bits of 4 bytes
                input_vect = ((encoding2 & mask) >> (i * 4)) << 3
```

```
                encoded_vect = input_vect
                generating_vect = 11
                tail = 8
                while input_vect >= 8:
                    while tail << 1 < input_vect:
                        tail = tail << 1
                        generating_vect = generating_vect << 1
                    input_vect = input_vect ^ generating_vect
                    tail = 8
                    generating_vect = 11
                encoded2 = encoded2 + ((encoded_vect + input_vect) << (i * 7))       #
4 bits encoded to 7 bits
                mask = mask << 4
            frame_encoded = bytes(encoded2.to_bytes(7,  byteorder='big')) +
frame_encoded
        frames_encoded.append(frame_encoded)
    i = 0
    """for frame in frames_encoded:
        i = i + len(frame)
    print('total send', i, ', frames:', len(frames_encoded))
    print('send', frames_encoded[0])"""
    #print('transmitting...', datetime.datetime.now())
    return frames_encoded


def receive(frames_encoded):
    check_table = {
        1: 0,
        2: 1,
        3: 3,
        4: 2,
        5: 6,
        6: 4,
        7: 5
    }
    frames = []
    #print('decoding...', datetime.datetime.now())

    #print('rec',len(frames_encoded))

    if not isinstance(frames_encoded, list):
        return
    #leng = 0
    for frame_encoded in frames_encoded:
        #leng = len(frame_encoded) + leng

        decoding1 = frame_encoded
        frame = bytes([])
# decoding1 - hole frame
        while len(decoding1) > 0:
            decoding2 = int.from_bytes(decoding1[len(decoding1) - 7:], byteorder='big')
# decoding2 - 7 bytes
            decoding1 = decoding1[:len(decoding1) - 7]
            mask = 127
            decoded2 = 0
            for i in range(8):                                          # encoding each 4
bits of 4 bytes
                input_vect = (decoding2 & mask) >> (i * 7)
                decoded_vect = input_vect
                generating_vect = 11
                tail = 8
                while input_vect >= 8:
```

```
                    while tail << 1 < input_vect:
                        tail = tail << 1
                        generating_vect = generating_vect << 1
                    input_vect = input_vect ^ generating_vect
                    tail = 8
                    generating_vect = 11
                if input_vect != 0:
                    decoded_vect = decoded_vect ^ (1 << check_table[input_vect])
                mask = mask << 7
                decoded2 = decoded2 + ((decoded_vect >> 3) << (i * 4))
            frame = bytes(decoded2.to_bytes(4, byteorder='big')) + frame
        #print(frame)
        while frame[0] == 0:
            frame = frame[1:]
        frames.append(frame)
    #print(len(frames[0]))
    #print('total length', leng, 'bytes', ', frames:', len(frames_encoded))
    """for frame in frames:
        print('rec', frame)"""
    #frames[0] = frames[0][3:]           # KOSTYL!!!!!!!!!!!!!!!!!111
    #print('recovering...', datetime.datetime.now())

    message_id = 0
    frame_amount = 0
    prim_frame = None
    isFile = False

    #print(frames[0])
    for frame in frames:            # finding primary frame
        if frame[0] > 63:
            message_id = frame[0] & 63
            frame_amount = frame[1] * 65536 + frame[2] * 256 + frame[3]
            prim_frame = frames.index(frame)
            if frames[prim_frame][4] > 127:
                isFile = True
            break
    if isFile:
        file_name = frames[prim_frame][5:frames[prim_frame][4]+6].decode()
        file_name = file_name[:file_name.find('?')]
        text = bytes([])
    else:
        text = frames[prim_frame][5:frames[prim_frame][4]+6]
    for i in range(frame_amount):
        text = text + frames[i + 1][5:frames[i + 1][4]+6]
    if not isFile:
        text = text.decode()
        print(text)

        plistener.line_recieved(text)

    #print('ready!', datetime.datetime.now())

    if isFile:
        receive_file(text, file_name)

        plistener.file_recieved(file_name)

    #threading.Timer(0.1, receive(phizical.ser_read())).start()


def send_file(file):
    f = open(file, 'rb')
```

```
    data = f.read()
    f.close()
    file_name = file[max(file.rfind("/"), file.rfind("\\"))+1:]
    #print(file_name)
    return send(data, True, file_name)


def receive_file(data, file_name):
    print('received file:', file_name)
    f = open('../downloads\\' + file_name, 'wb')
    f.write(data)
    f.close()
    return file_name


def send_nudes(nudes):
    pass


def listener_connection(state):
    plistener.connecting(state)


def listener_user_connection(state):
    plistener.user_connecting(state)


def listener_transmission_failed():
    plistener.transmission_failed()
```

parser.py:

```
import json
from sys import platform

# ------------------------------------------------------------------
# Функция парсит config.json и возвращает значения параметров
# открываемых COM-портов
# Используется в phizical.portinit(com_params)
# ------------------------------------------------------------------


def get_port_parameters():
    with open('../config.json', 'r') as fp:
        obj = json.load(fp)

        if platform == "win32":
            first_COM = obj["win_port1"]
            second_COM = obj["win_port2"]
        else:
            first_COM = obj["linux_port1"]
            second_COM = obj["linux_port2"]

        baudrate = obj["baudrate"]
        timeout = obj["timeout"]
        writeTimeout = obj["writeTimeout"]

    return first_COM, second_COM, baudrate, timeout, writeTimeout
```

listener.py:

```
import datetime
from PyQt5 import QtCore

# ----------------------- PortListener -------------------------------
# ------------------- класс прослушки портов -------------------------
# --------------------------------------------------------------------
# line                | объекты, передаваемые в главное окно ChatApp
# file                | когда channel.recieve() отдает сообщение/файл
# connected           | content = сообщение / text = название файла
# user_connected      | connected = разрыв связи / связь поддерживается
#                     | user_connected = соединение с другим пользова-
#                     | телем
# transmission_error  | transmission_error = ошибка передачи файла
# --------------------|-----------------------------------------------
# line_recieved       | функции-триггеры
# file_recieved       | line = show_message / file = show_file
# connecting          | функции, проверяющие состояние подключения поль-
# user_connecting     | зователей
#                     | 0 - нет подключения, 1 - соединение установлено
# transmission_failed | функция-триггер ошибки передачи
#
# Определен в channels.py и связан с ChatApp
# --------------------------------------------------------------------


class PortListener(QtCore.QObject):
    line = QtCore.pyqtSignal(str)
    file = QtCore.pyqtSignal(str)
    connected = QtCore.pyqtSignal(bool)
    user_connected = QtCore.pyqtSignal(bool)
    transmission_error = QtCore.pyqtSignal(str)

    def line_recieved(self, text):
        content = f"{datetime.datetime.now().strftime('%H:%M')} <Собеседник>: {text}"
        self.line.emit(content)

    def file_recieved(self, text):
        self.file.emit(text)

    def connecting(self, state):
        if state == 0:
            self.connected.emit(state)

    def user_connecting(self, state):
        self.user_connected.emit(state)

    def transmission_failed(self):
        content = "Произошла ошибка. Сообщение не было доставлено"
        self.transmission_error.emit(content)
```

info dialog.py:

```
from PyQt5 import QtCore, QtWidgets
from gui import info
```

```
# ---------------------- InfoDialog -------------------------
# --- класс модального окна | gui/info.py, gui/ui/info.ui ---
# ----------------------------------------------------------
# bOK               кнопка OK (закрыть диалоговое окно)
# setFixedSize      запрет на растягивание окна
# Вызывается из ChatApp по кнопке тулбара Информация
# ----------------------------------------------------------


class InfoDialog(QtWidgets.QDialog, info.Ui_dialog):
    def __init__(self, parent=None):
        super().__init__(parent, QtCore.Qt.Dialog)
        self.setupUi(self)
        self.init_handlers()
        self.setFixedSize(self.size())

    def init_handlers(self):
        self.bOK.clicked.connect(self.close)
```

info.py:

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'gui/info.ui'
#
# Created by: PyQt5 UI code generator 5.14.2
#
# WARNING! All changes made in this file will be lost!


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_dialog(object):
    def setupUi(self, dialog):
        dialog.setObjectName("dialog")
        dialog.resize(416, 300)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(dialog.sizePolicy().hasHeightForWidth())
        dialog.setSizePolicy(sizePolicy)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("../gui/icon/info.svg"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        dialog.setWindowIcon(icon)
        dialog.setModal(True)
        self.verticalLayoutWidget = QtWidgets.QWidget(dialog)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(9, 9, 401, 281))
        self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.label = QtWidgets.QLabel(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily("Helvetica")
        font.setPointSize(10)
        font.setItalic(False)
```

```
        self.label.setFont(font)
        self.label.setAutoFillBackground(False)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setObjectName("label")
        self.verticalLayout.addWidget(self.label)
        self.bOK = QtWidgets.QPushButton(self.verticalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.bOK.sizePolicy().hasHeightForWidth())
        self.bOK.setSizePolicy(sizePolicy)
        self.bOK.setMinimumSize(QtCore.QSize(120, 40))
        self.bOK.setObjectName("bOK")
        self.verticalLayout.addWidget(self.bOK, 0, QtCore.Qt.AlignHCenter)

        self.retranslateUi(dialog)
        QtCore.QMetaObject.connectSlotsByName(dialog)

    def retranslateUi(self, dialog):
        _translate = QtCore.QCoreApplication.translate
        dialog.setWindowTitle(_translate("dialog", "Информация"))
        self.label.setText(_translate("dialog", "Курсовая работа\n"
"по курсу \"Сетевые технологии в АСОИУ\"\n"
"на тему \"Локальная безадаптерная сеть\"\n"
"вариант №6\n"
"\n"
"Исполнители:\n"
" Болгова А.В., Фонканц Р.В., Попов И.А.\n"
"\n"
"\n"
"Москва, 2020"))
        self.bOK.setText(_translate("dialog", "OK"))
```

window.py:

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'gui/window.ui'
#
# Created by: PyQt5 UI code generator 5.14.2
#
# WARNING! All changes made in this file will be lost!


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(906, 714)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("../gui/icon/chat.svg"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        MainWindow.setWindowIcon(icon)
        MainWindow.setIconSize(QtCore.QSize(32, 32))

        self.layout_toolbar = QtWidgets.QHBoxLayout()
        self.layout_toolbar.setObjectName("layout_toolbar")
```

```
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.centralwidget.sizePolicy().hasHeightForWidth())
        self.centralwidget.setSizePolicy(sizePolicy)
        self.centralwidget.setObjectName("centralwidget")

        self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.centralwidget)
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")

        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setSizeConstraint(QtWidgets.QLayout.SetMinimumSize)
        self.verticalLayout.setObjectName("verticalLayout")

        self.verticalLayout.addLayout(self.layout_toolbar)

        self.textList = QtWidgets.QListWidget(self.centralwidget)
        self.textList.setWordWrap(True)
        self.textList.setSelectionMode(QtWidgets.QAbstractItemView.NoSelection)
        self.textList.setFont(QtGui.QFont('Helvetica', 10))
        self.textList.setSpacing(5)
        self.textList.setObjectName("textList")
        self.verticalLayout.addWidget(self.textList)

        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")

        self.message = QtWidgets.QLineEdit(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Maximum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.message.sizePolicy().hasHeightForWidth())
        self.message.setSizePolicy(sizePolicy)
        self.message.setMaximumSize(QtCore.QSize(16777215, 100))
        self.message.setFont(QtGui.QFont('Helvetica', 10))
        self.message.setAlignment(QtCore.Qt.AlignTop)
        self.message.setObjectName("message")
        self.horizontalLayout.addWidget(self.message)

        self.verticalLayout_2 = QtWidgets.QVBoxLayout()
        self.verticalLayout_2.setSizeConstraint(QtWidgets.QLayout.SetDefaultConstraint)
        self.verticalLayout_2.setSpacing(6)
        self.verticalLayout_2.setObjectName("verticalLayout_2")

        self.bSend = QtWidgets.QPushButton(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.bSend.sizePolicy().hasHeightForWidth())
        self.bSend.setSizePolicy(sizePolicy)
        self.bSend.setMinimumSize(QtCore.QSize(150, 40))
        self.bSend.setObjectName("bSend")
        self.verticalLayout_2.addWidget(self.bSend)

        self.bSendFile = QtWidgets.QPushButton(self.centralwidget)
        self.bSendFile.setMinimumSize(QtCore.QSize(150, 40))
```

```
        self.bSendFile.setObjectName("bSendFile")
        self.verticalLayout_2.addWidget(self.bSendFile)

        self.horizontalLayout.addLayout(self.verticalLayout_2)
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.horizontalLayout_2.addLayout(self.verticalLayout)
        MainWindow.setCentralWidget(self.centralwidget)

        self.statusBar = QtWidgets.QStatusBar(MainWindow)
        self.statusBar.setObjectName("statusBar")
        MainWindow.setStatusBar(self.statusBar)

        self.toolBar = QtWidgets.QToolBar(MainWindow)
        self.toolBar.setObjectName("toolBar")
        MainWindow.addToolBar(QtCore.Qt.TopToolBarArea, self.toolBar)

        self.mConnect = QtWidgets.QAction(MainWindow)
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap("../gui/icon/connected.png"), QtGui.QIcon.Normal,
QtGui.QIcon.On)
        self.mConnect.setIcon(icon1)
        self.mConnect.setObjectName("mConnect")

        self.mDisonnect = QtWidgets.QAction(MainWindow)
        icon4 = QtGui.QIcon()
        icon4.addPixmap(QtGui.QPixmap("../gui/icon/disconnected.png"),
QtGui.QIcon.Normal, QtGui.QIcon.On)
        self.mDisonnect.setIcon(icon4)
        self.mDisonnect.setObjectName("mDisonnect")

        self.mExit = QtWidgets.QAction(MainWindow)
        icon2 = QtGui.QIcon()
        icon2.addPixmap(QtGui.QPixmap("../gui/icon/exit.svg"), QtGui.QIcon.Normal,
QtGui.QIcon.On)
        self.mExit.setIcon(icon2)
        self.mExit.setObjectName("mExit")

        self.mInfo = QtWidgets.QAction(MainWindow)
        icon3 = QtGui.QIcon()
        icon3.addPixmap(QtGui.QPixmap("../gui/icon/info.svg"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        self.mInfo.setIcon(icon3)
        self.mInfo.setObjectName("mInfo")

        self.statusLabel = QtWidgets.QLabel(self.centralwidget)
        self.statusLabel.setScaledContents(True)
        self.statusLabel.setFixedSize(24, 24)
        self.statusLabel.setAlignment(QtCore.Qt.AlignRight)

        self.pixmap = QtGui.QPixmap("../gui/icon/offline.png")
        self.statusLabel.setPixmap(self.pixmap)

        self.toolBar.addAction(self.mConnect)
        self.toolBar.addAction(self.mDisonnect)
        self.toolBar.addAction(self.mExit)
        self.toolBar.addSeparator()
        self.toolBar.addAction(self.mInfo)

        self.layout_toolbar.addWidget(self.toolBar)
        self.layout_toolbar.addWidget(self.statusLabel)
        self.layout_toolbar.setContentsMargins(0, 0, 10, 0)
```

```
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Главное окно"))
        self.bSend.setText(_translate("MainWindow", "Отправить"))
        self.bSendFile.setText(_translate("MainWindow", "Прикрепить файл"))
        self.toolBar.setWindowTitle(_translate("MainWindow", "toolBar"))
        self.mConnect.setText(_translate("MainWindow", "Подключиться"))
        self.mExit.setText(_translate("MainWindow", "Выход"))
        self.mInfo.setText(_translate("MainWindow", "Информация"))
        self.statusLabel.setToolTip(_translate("MainWindow", "Статус собеседника"))
```