МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

ОТЧЕТ

по лабораторной работе № 6
на тему: «Сетевое программирование»
по дисциплине: «Программирование на языке Python»
Вариант № 18

Выполнил: Шорин В.Д. Шифр: 171406
Институт приборостроения, автоматизации и информационных технологий
Направление: 09.03.04 «Программная инженерия»
Группа: 71-ПГ
Проверили: Захарова О.В., Раков В.И.

Отметка о зачете:

Дата: «____» _____ 2019 г.

Орел, 2019 г.

Задание:

Разработать клиент-серверное приложение (чат).

Требования к клиенту:

- отправка на сервер введенного пользователем сообщения;

- получение сообщений, отправленных другими клиентами;

- удобный графический интерфейс.

Требования к серверу:

- организация чата между клиентами (до 3 клиентов), а именно отправка всем клиентам полученных сообщений;

- полученные сообщения перед отправкой клиентам необходимо проверить на грамотность и исправить: после запятой должен стоять один пробел, далее должен идти непробельный символ.

Клиенты должны последовательно отправлять сообщения.

Код:

«main.py» Client

```
import tkinter
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread


def receive():
    """ Handles receiving of messages. """
    while True:
        try:
            msg = sock.recv(BUFSIZ).decode("utf8")
            msg_list.insert(tkinter.END, msg)
        except OSError:  # Possibly client has left the chat.
            break


def send(event=None):
    """ Handles sending of messages. """
    msg = my_msg.get()
    my_msg.set("")  # Clears input field.
    sock.send(bytes(msg, "utf8"))
    if msg == "#quit":
        sock.close()
        top.quit()
```

```
def on_closing(event=None):
    """ This function is to be called when the window is closed. """
    my_msg.set("#quit")
    send()


top = tkinter.Tk()
top.title("Simple Chat Client v1.0")
messages_frame = tkinter.Frame(top)

my_msg = tkinter.StringVar()  # For the messages to be sent.
my_msg.set("")
scrollbar = tkinter.Scrollbar(messages_frame)  # To navigate through past
messages.
msg_list = tkinter.Listbox(messages_frame, height=15, width=70,
yscrollcommand=scrollbar.set)
scrollbar.pack(side=tkinter.RIGHT, fill=tkinter.Y)
msg_list.pack(side=tkinter.LEFT, fill=tkinter.BOTH)
msg_list.pack()

messages_frame.pack()

button_label = tkinter.Label(top, text="Enter Message:")
button_label.pack()

entry_field = tkinter.Entry(top, textvariable=my_msg, foreground="Red")
entry_field.bind("<Return>", send)
entry_field.pack()

send_button = tkinter.Button(top, text="Send", command=send)
send_button.pack()


quit_button = tkinter.Button(top, text="Quit", command=on_closing)
quit_button.pack()

top.protocol("WM_DELETE_WINDOW", on_closing)

HOST = "127.0.0.1"
PORT = 5000
BUFSIZ = 1024
ADDR = (HOST, PORT)
sock = socket(AF_INET, SOCK_STREAM)
sock.connect(ADDR)
```

```
receive_thread = Thread(target=receive)
receive_thread.start()
tkinter.mainloop()  # Starts GUI execution.
```

«processor.py» Server

```
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread
import re

clients = {}
addresses = {}

HOST = "127.0.0.1"
PORT = 5000
BUFSIZ = 1024
ADDR = (HOST, PORT)
SOCK = socket(AF_INET, SOCK_STREAM)
SOCK.bind(ADDR)


def parse_message(msg):
    message = msg.decode()
    # pattern_whirespace = '[\s]+'
    # re.sub(pattern_whirespace, ' ', message)
    i = 0
    while i != len(message) - 3:
        if message[i] == ',' \
                and message[i + 1] != ' ':
            message = message[:i] + ', ' + message[i + 1:]
            i = 0
        elif message[i] == ',' \
                and message[i + 1] == ' ' \
                and message[i + 2] == ' ':
            message = message[:i] + ', ' + message[i + 3:]
            i = 0
        else:
            message = message
            i += 1

    return message.encode()


def accept_incoming_connections():
    """Sets up handling for incoming clients."""
    while True:
```

```
        client, client_address = SOCK.accept()
        print(f"{client_address} has connected.")
        client.send("Greetings from the ChatRoom! ".encode("utf8"))
        client.send("Now type your name and press enter!".encode("utf8"))

        addresses[client] = client_address
        Thread(target=handle_client, args=(client, client_address)).start()


def handle_client(conn, addr):  # Takes client socket as argument.
    """Handles a single client connection."""
    name = conn.recv(BUFSIZ).decode("utf8")

    welcome = f'Welcome {name}! If you ever want to quit, type #quit to exit.'
    conn.send(bytes(welcome, "utf8"))

    msg = f"{name} from [{addr[0]}:{addr[1]}] has joined the chat!"
    broadcast(bytes(msg, "utf8"))

    clients[conn] = name
    while True:
        msg = conn.recv(BUFSIZ)
        msg = parse_message(msg)
        if msg != bytes("#quit", "utf8"):
            broadcast(msg, name + ": ")
        else:
            conn.send(bytes("#quit", "utf8"))

            conn.close()
            del clients[conn]

            broadcast(bytes(f"{name} has left the chat.", "utf8"))
            break


def broadcast(msg, prefix=""):  # prefix is for name identification.
    """Broadcasts a message to all the clients."""
    for sock in clients:
        sock.send(bytes(prefix, "utf8") + msg)


if __name__ == "__main__":
    SOCK.listen(3)  # Listens for 3 connections at max.

    print("Chat Server has Started !!")
```

```
print("Waiting for connections...")

ACCEPT_THREAD = Thread(target=accept_incoming_connections)
ACCEPT_THREAD.start()  # Starts the infinite loop.
ACCEPT_THREAD.join()
SOCK.close()
```