ФГБОУ ВПО

Уфимский университет науки и технологий

Отчет

по лабораторной работе №3

по дисциплине «Архитектура вычислительных систем»

Тема: «Основы сетевого программирования. Многопоточные обработка сокет соединений»

Выполнили: ст. гр. ПРО-331Б

Семенов Л.А.

Проверил:

Юдинцев Б.C.

Уфа, 2024 г

**Задание**

1. Изучить примеры кода с реализацией многопоточного сокет-сервера и

клиента (см. файлы в архиве lab3\_ПРИМЕР.zip).

2. Добавить в сокет-сервер следующие функции:

- проверка уникальности имени клиента при подключении (если

имя повторяется, то отправить клиенту соответствующее

сообщение об ошибке).

- Поименная адресация сообщений (если пользователь не найден, то

вывод сообщения об ошибке).

- Добавить в класс сокет-соединения функцию записи всех

сообщений пользователей в общий log-файл.

3. Запустить сокет-сервер на ВМ (должна быть создана в процессе

выполнения лабораторной работы №2). Проверить обмен сообщениями

между несколькими клиентскими приложениями через сокет-сервер,

приложить скриншоты (демонстрирующие работу добавленных функций) к

отчету.

Ход работы:

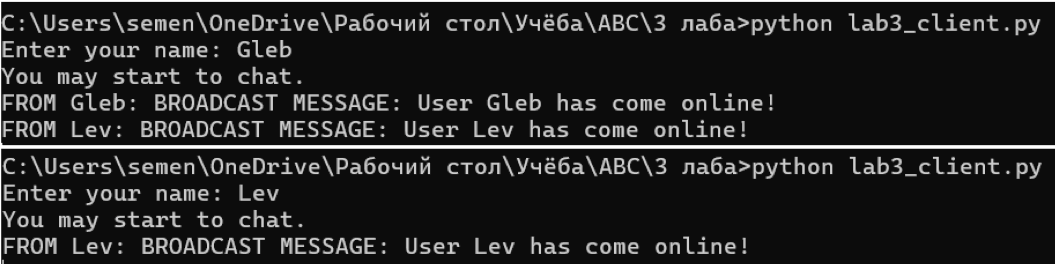


Рисунок 1. Регистрация

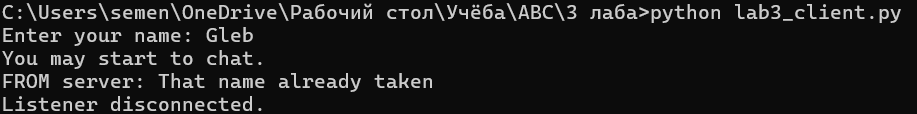


Рисунок 2. Проверка на уникальность имени

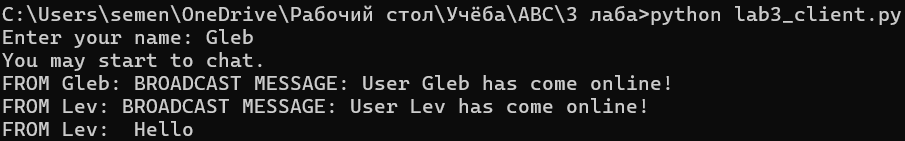


Рисунок 3. Отправка персональных сообщений

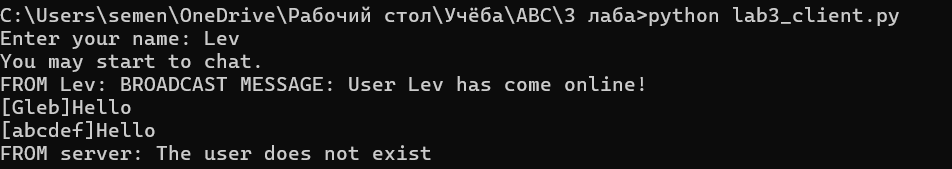


Рисунок 4. Ввод несуществующего имени

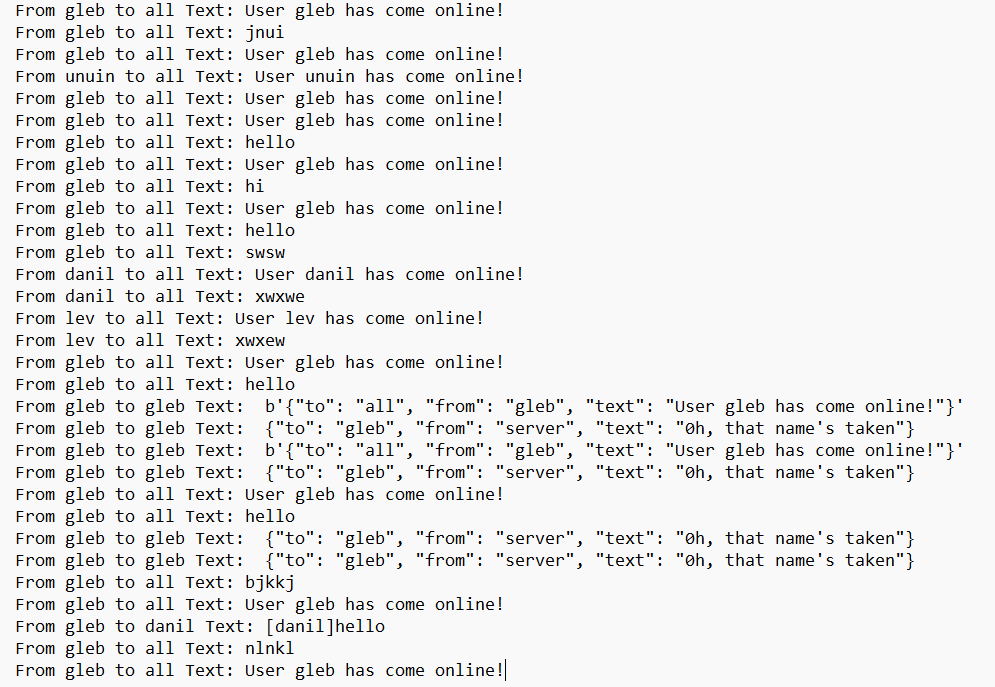


Рисунок 5. Лог-файл сообщений

**Вывод**

В ходе лабораторной работы были реализованы функции отправки персональных сообщений, проверка имен на уникальность и логирование файлов

**Приложение 1**

Файл connthread.py

import threading  
from threading import Thread  
import json  
  
CONNECTION\_POOL = [] # список для хранения объектов потоков  
LOCK = threading.Lock()  
  
"""  
 В данном модуле реализуется класс для работы с сокет соединением в отдельном потоке  
"""  
  
# Наследуем класс от базового класса Thread  
class SocketConnectionThread(Thread):  
 username = None # Имя подключенного пользователя  
 # В конструкторе класса определяем внутренние атрибуты для хранения адреса клиента (self.address),  
 # объекта соединения (self.connection) и имени клиента (username)  
 def \_\_init\_\_(self, conn, addr):  
 # вызываем конструктор родительского класса  
 Thread.\_\_init\_\_(self)  
 self.connection = conn  
 self.address = addr  
  
 # Переопределяем метод run родительского класса Thread  
 def run(self):  
 # запускаем ожидание команд от клиента в бесконечном цикле  
 while True:  
 try:  
 data = self.connection.recv(1024) # ожидаем данные от клиента (максимум 1024 байт)  
 data\_json = json.loads(data.decode('utf-8')) # преобразуем полученные от клиента байты в json-строку  
 data\_str = data\_json['text'] # получаем текст сообщения  
 data\_from = data\_json['from'] # получаем имя создателя сообщения  
 # Если имя клиента не назначено (поток только что создан)  
 if self.username is None:  
 self.username = data\_from # Назначаем имя пользователя для соединения  
 if self.check\_name(self.username):  
 break  
 data\_to = data\_json['to']  
 print(f"FROM {data\_from} TO {data\_to}: {data\_str}") # вывод полученных от клиента данных  
  
 # Если от клиента получена команда "exit" или "close", закрываем соединение  
 if data\_str == "exit" or data\_str == "close":  
 print(f"User {data\_from} has gone offline!") # Всем сообщаем об отключении пользователя  
 break # прерываем цикл  
 # Если адресат не назначен или 'all', то отправляем широковещательное сообщение  
 elif data\_to == '' or data\_to == 'all':  
 if data\_str != '':  
 self.send\_all(data\_str)  
 self.log\_write(data)  
 else:  
 self.send\_to(data\_str, data\_to)  
 self.log\_write(data)  
 response = f"SERVER RESPONSE: MESSAGE RECEIVED FROM {data\_from}" # сообщение об успешном получении данных от клиента  
 print(response)  
 except Exception as error:  
 print(self.name, error)  
 break  
 self.connection.close() # закрываем соединение  
 self.remove\_connection\_from\_pool() # удаляем объект соединения из общего пула CONNECTION\_POOL  
  
 # Широковещательная отправка сообщения  
 def send\_all(self, text):  
 msg = json.dumps({'to': "all", 'from': self.username, 'text': f"BROADCAST MESSAGE: {text}"})  
 for thread in CONNECTION\_POOL:  
 thread.connection.send(msg.encode('utf-8'))  
  
 def check\_name(self, name):  
 i = 0  
 for thread in CONNECTION\_POOL:  
 if thread.username == name:  
 i = i + 1  
 if i == 2:  
 msg = json.dumps({'to': self.username, 'from': 'server', 'text': 'That name already taken'})  
 self.connection.send(msg.encode('utf-8'))  
 return True  
 return False  
 def send\_to(self, text, to):  
 index = text.find("]")  
 formatedText = text[index + 1:]  
 msg = json.dumps({'to': to, 'from': self.username, 'text': f" {formatedText}"})  
 check = True  
 for thread in CONNECTION\_POOL: