МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет по лабораторной работе № 12**

по дисциплине: “Системное программирование”

# на тему: *“* Клиент-серверное сокетное соединение в сети. *”*

Вариант 1

Выполнил**:** студент группы 10702121 Меркулова М. С.

Черняк П. С.

Принял**:** Хавитько Е. А.

Минск 2023

# Лабораторная работа Клиент-серверное сокетное соединение в сети.

Цель работы: Изучить механизм сокетного сетевого взаимодействия в LINUX.

**Задание 1**

Напишите программы с клиент-серверной архитектурой по следующим тре- бованиям:

1. Обменяться числами. Сервер возводит получаемое от клиента число в квадрат и возвращает ответ клиенту.

2. Создать два клиента на один и тот же сокет. Используют один и тот же сокет. Первый клиент получает результат счета от 1 до 10, второй – ре- зультат счета от 100 до 110 (подумайте, как распознать, какой клиент обратился к серверу).

Создать два клиента на один и тот же сервер. Сокеты должны быть раз- ными (именованное соединение в каждом случае должно быть различ- ным). Первый клиент получает результат счета от 1 до 10, второй – от 100 до 110. Число процессов-клиентов и сокетов должно соответствовать числу 2

**Задание 2**

Напишите программу, которая читает главную страницу из какого-нибудь Web-сервера. Запуск программы должен производиться с консоли.

**Варианты**, *простые числа* – подключиться к странице сайта БНТУ, ФИТР; *четные числа* – подключиться к странице сайта БНТУ, расписание занятий 3-го курса; *кратные трем* – подключиться к странице сайта БНТУ, абитуриент, по- ступление.

**Задание 3**

Напишите программу с клиент-серверной архитектурой для передачи данных через дейтаграммные сокеты. Количество клиентов должно соответствовать количеству членов вашей команды, но не менее двух. Семейства протоколов и способы взаимодействия сокетов-клиентов должны отличаться.

Запустите программы в разных терминалах и определите какие программы- клиенты получили данные. Объясните полученный результат.

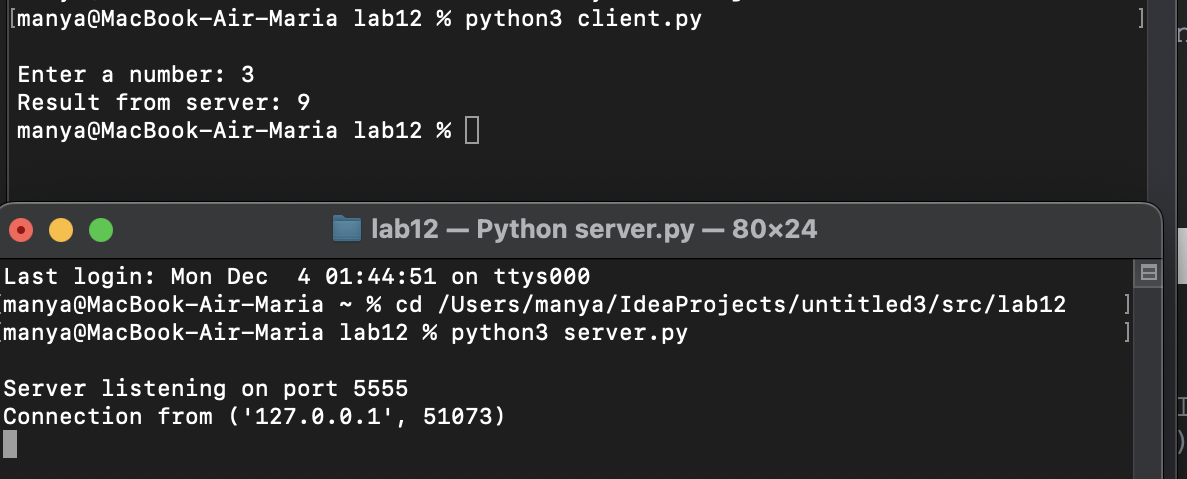
**Решение**

**Задание 1**

**Пример 1: Обмен числами, сервер возводит число в квадрат и возвращает ответ клиенту.**

import socket  
  
def square\_number(number):  
 return number \*\* 2  
  
def main():  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind(('127.0.0.1', 5555))  
 server\_socket.listen(1)  
 print("Server listening on port 5555")  
  
 while True:  
 client\_socket, client\_address = server\_socket.accept()  
 print(f"Connection from {client\_address}")  
  
 data = client\_socket.recv(1024).decode()  
 if data.isdigit():  
 number = int(data)  
 result = square\_number(number)  
 client\_socket.send(str(result).encode())  
  
 client\_socket.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

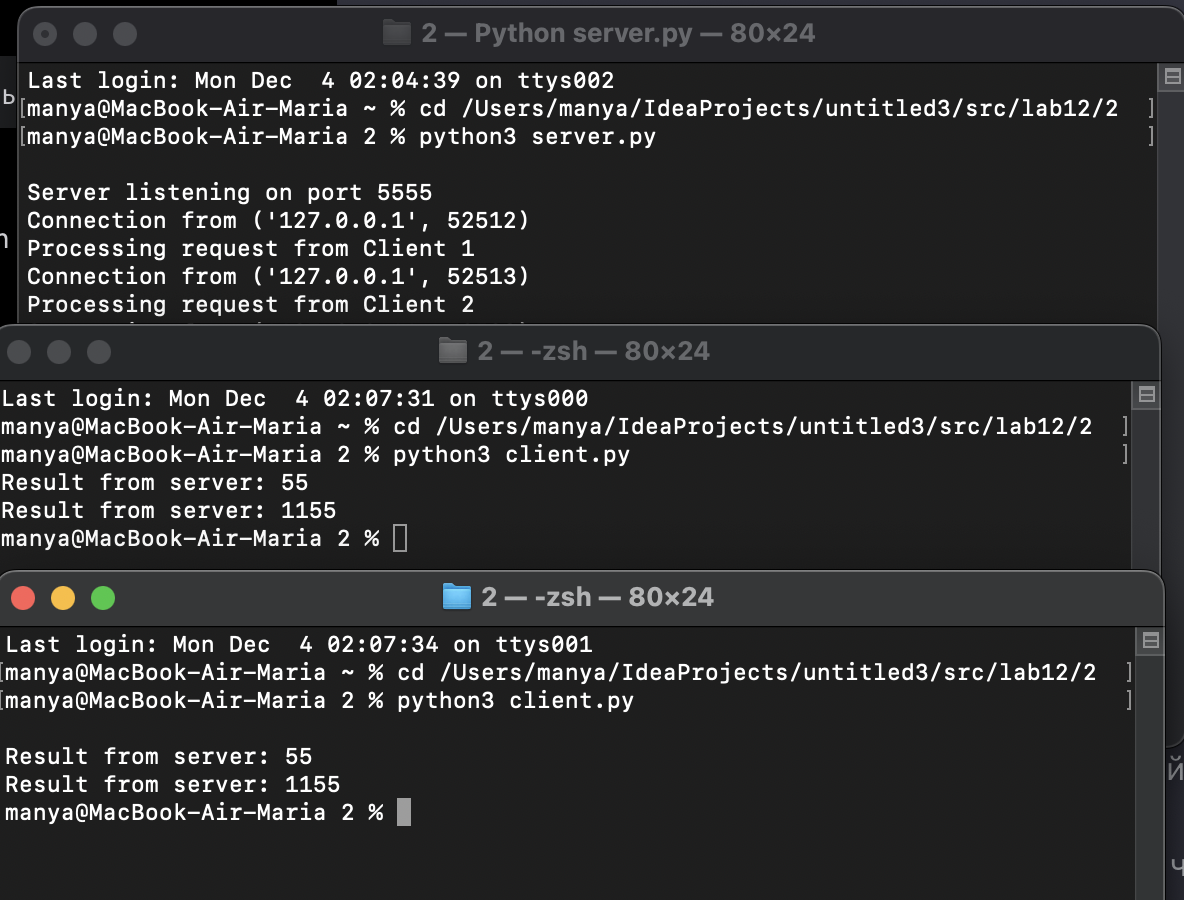
import socket  
  
def main():  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(('127.0.0.1', 5555))  
  
 number = int(input("Enter a number: "))  
 client\_socket.send(str(number).encode())  
  
 result = client\_socket.recv(1024).decode()  
 print(f"Result from server: {result}")  
  
 client\_socket.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

****

**Пример 2: Создание двух клиентов на один сокет, различие по числовому диапазону.**

import socket  
  
def main(start, end):  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(('127.0.0.1', 5555))  
  
 data = f"{start}-{end}"  
 client\_socket.send(data.encode())  
  
 result = client\_socket.recv(1024).decode()  
 print(f"Result from server: {result}")  
  
 client\_socket.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main(1, 10) # Первый клиент, диапазон от 1 до 10  
 main(100, 110) # Второй клиент, диапазон от 100 до 110

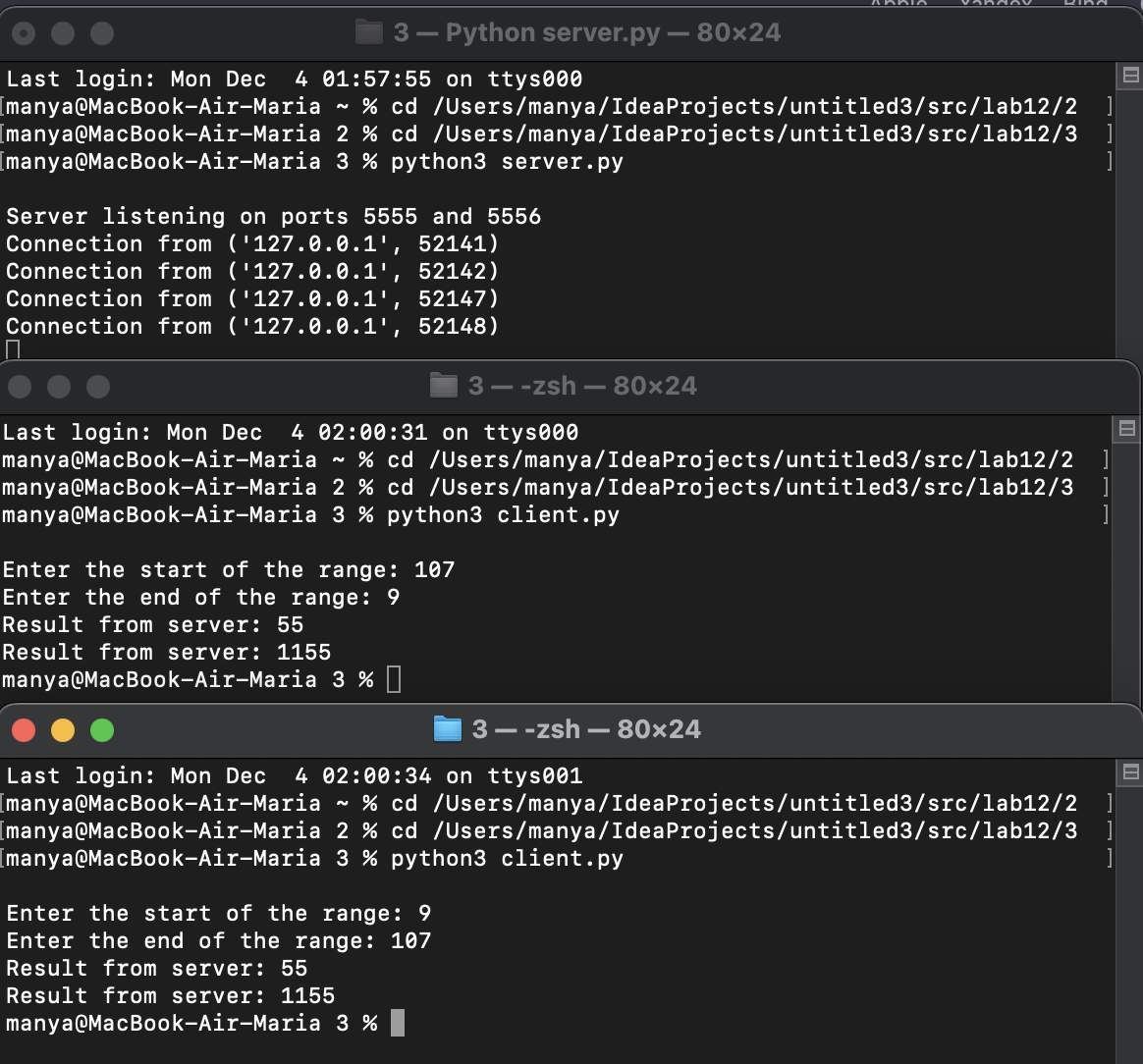
import socket  
  
def calculate\_result(start, end):  
 result = sum(range(start, end + 1))  
 return result  
  
def main():  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind(('127.0.0.1', 5555))  
 server\_socket.listen(2)  
 print("Server listening on port 5555")  
  
 while True:  
 client\_socket, client\_address = server\_socket.accept()  
 print(f"Connection from {client\_address}")  
  
 data = client\_socket.recv(1024).decode()  
 start, end = map(int, data.split('-'))  
 result = calculate\_result(start, end)  
 client\_socket.send(str(result).encode())  
  
 client\_socket.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

****

**Пример 3: Создание двух клиентов на разных сокетах.**

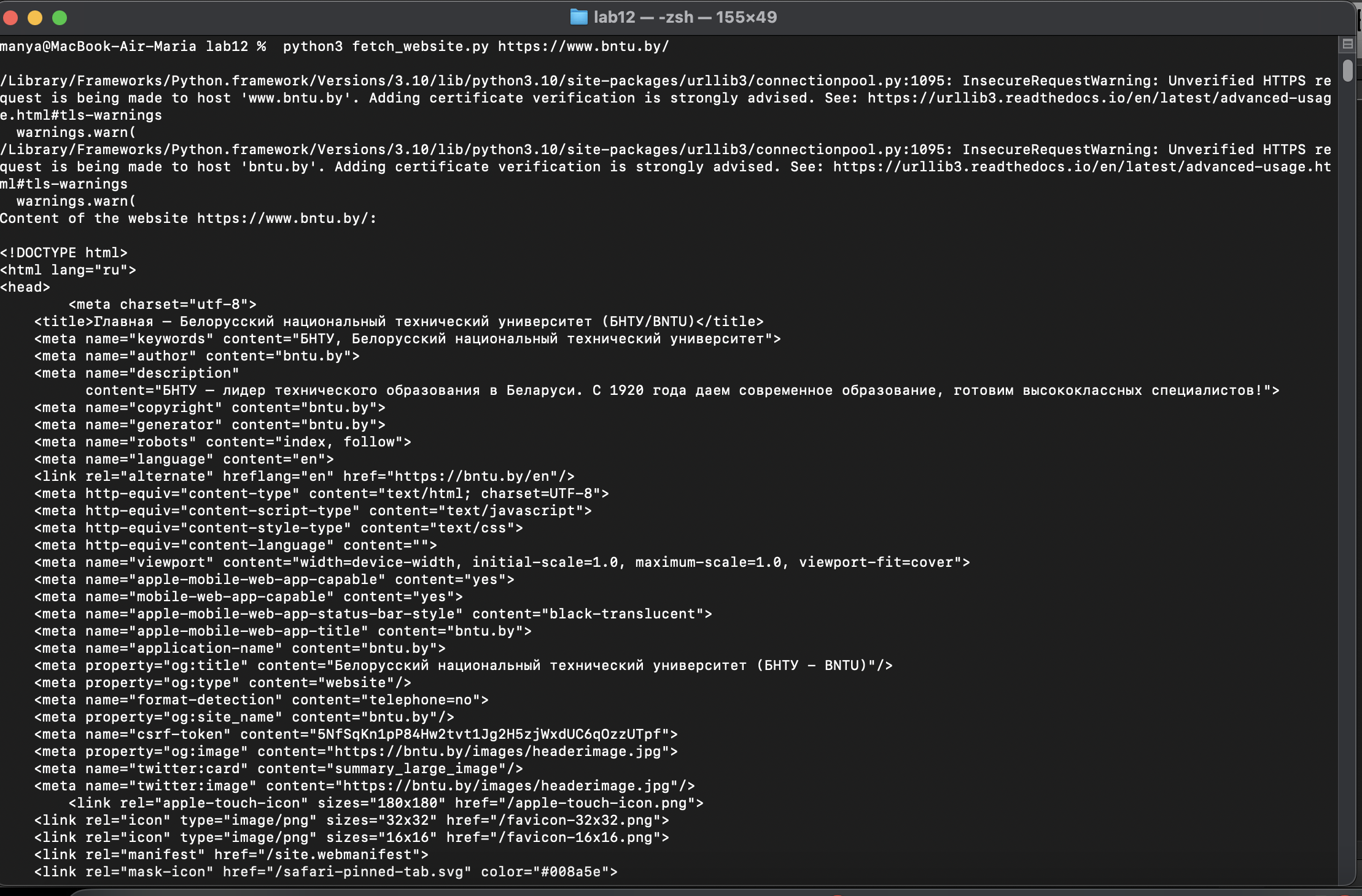
import socket  
import threading  
  
def calculate\_result(start, end):  
 result = sum(range(start, end + 1))  
 return result  
  
def handle\_client(client\_socket, start, end):  
 data = client\_socket.recv(1024).decode()  
 result = calculate\_result(start, end)  
 client\_socket.send(str(result).encode())  
 client\_socket.close()  
  
def main():  
 server\_socket1 = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket1.bind(('127.0.0.1', 5555))  
 server\_socket1.listen(1)  
  
 server\_socket2 = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket2.bind(('127.0.0.1', 5556))  
 server\_socket2.listen(1)  
  
 print("Server listening on ports 5555 and 5556")  
  
 while True:  
 client\_socket1, client\_address1 = server\_socket1.accept()  
 print(f"Connection from {client\_address1}")  
 threading.Thread(target=handle\_client, args=(client\_socket1, 1, 10)).start()  
  
 client\_socket2, client\_address2 = server\_socket2.accept()  
 print(f"Connection from {client\_address2}")  
 threading.Thread(target=handle\_client, args=(client\_socket2, 100, 110)).start()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

import socket  
  
def main(port, start, end):  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(('127.0.0.1', port))  
  
 data = f"{start}-{end}"  
 client\_socket.send(data.encode())  
  
 result = client\_socket.recv(1024).decode()  
 print(f"Result from server: {result}")  
  
 client\_socket.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main(5555, 1, 10) # Первый клиент, диапазон от 1 до 10  
 main(5556, 100, 110) # Второй клиент, диапазон от 100 до 110

****

**Задание 2**

import requests  
import sys  
  
def fetch\_website(url):  
 try:  
 response = requests.get(url, verify=False)  
 response.raise\_for\_status() # Проверка на ошибки в запросе  
 return response.text  
 except requests.RequestException as e:  
 print(f"Error: {e}")  
 sys.exit(1)  
  
def main():  
 if len(sys.argv) != 2:  
 print("Usage: python fetch\_website.py <url>")  
 sys.exit(1)  
  
 url = sys.argv[1]  
 website\_content = fetch\_website(url)  
  
 print(f"Content of the website {url}:\n")  
 print(website\_content)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

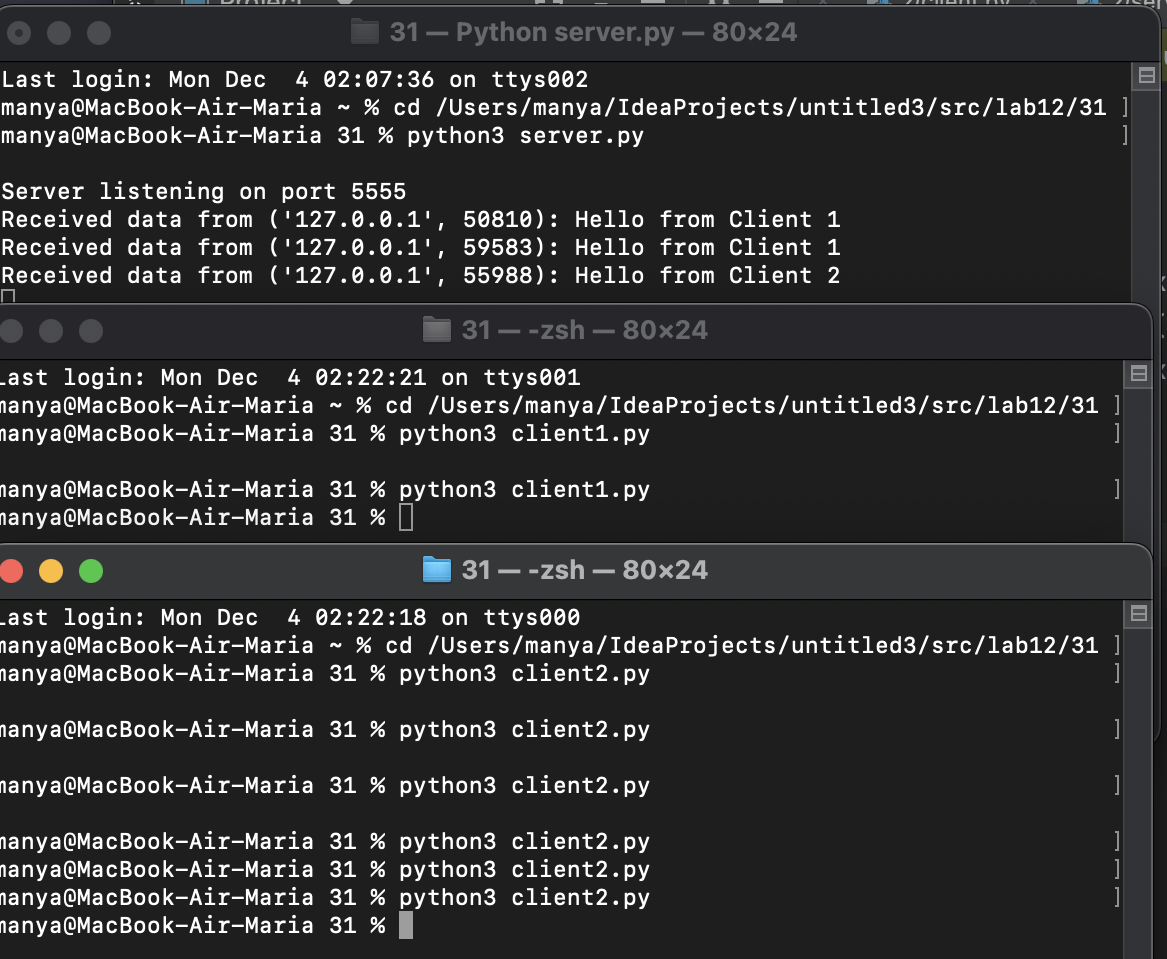
****

**Задание 3**

* server.py создает дейтаграммный сокет и прослушивает порт 5555. Он ожидает данные от любого клиента и выводит их в консоль.
* client1.py создает дейтаграммный сокет с семейством протоколов IPv4 и отправляет строку "Hello from Client 1" на порт 5555 сервера.
* client2.py создает дейтаграммный сокет с семейством протоколов IPv6 и отправляет строку "Hello from Client 2" на порт 5555 сервера.
* import socket  
    
  def main():  
   server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
   server\_socket.bind(('127.0.0.1', 5555))  
    
   print("Server listening on port 5555")  
    
   while True:  
   data, client\_address = server\_socket.recvfrom(1024)  
   print(f"Received data from {client\_address}: {data.decode()}")  
    
  if \_\_name\_\_ == "\_\_main\_\_":  
   main()

import socket  
  
def main():  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 client\_socket.sendto(b"Hello from Client 1", ('127.0.0.1', 5555))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

import socket  
  
def main():  
 client\_socket = socket.socket(socket.AF\_INET6, socket.SOCK\_DGRAM)  
 client\_socket.sendto(b"Hello from Client 2", ('::1', 5555))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()



**Объяснение результатов:**

* Сервер ожидает данные на порту 5555 и принимает их от любого клиента, используя дейтаграммные сокеты.
* **client1.py** отправляет данные через IPv4, используя сокет с семейством протоколов IPv4.
* **client2.py** отправляет данные через IPv6, используя сокет с семейством протоколов IPv6.

В результате каждый клиент может успешно отправить данные на сервер, и сервер может успешно принять их, даже если сокеты клиентов используют разные семейства протоколов.

**Контрольный вопрос**

1. Что такое удаленное межпроцессное взаимодействие?

Удаленное межпроцессное взаимодействие (Remote Inter-Process Communication, Remote IPC) — это процесс обмена данными и командами между процессами, выполняющимися на удаленных устройствах или в удаленных сетях. В отличие от межпроцессного взаимодействия (IPC) на локальной машине, где процессы взаимодействуют в пределах одного устройства, удаленное IPC позволяет процессам взаимодействовать через сеть, преодолевая границы между физическими устройствами или даже различными распределенными системами.

Примеры удаленного IPC включают в себя:

1. **Сетевые сокеты:** Процессы могут использовать сокеты для обмена данными через сеть. Протоколы, такие как TCP/IP или UDP, позволяют процессам на удаленных устройствах общаться друг с другом.
2. **RPC (Remote Procedure Call):** RPC предоставляет механизм вызова удаленных процедур или функций. Клиентский процесс вызывает процедуру на удаленном сервере так, как если бы она была локальной.
3. **RESTful API:** RESTful API (Representational State Transfer) обеспечивает удаленное взаимодействие между клиентами и серверами посредством использования стандартных HTTP-методов, таких как GET, POST, PUT, DELETE.
4. **Message Queues:** Удаленные процессы могут взаимодействовать через системы сообщений, такие как RabbitMQ, Apache Kafka и другие. Они позволяют отправлять и получать сообщения между процессами, даже если они выполняются на разных устройствах.

Удаленное IPC имеет много применений, особенно в распределенных системах, облачных вычислениях и микросервисной архитектуре, где части приложения могут быть развернуты на различных серверах или в облаке.