**Лабораторна робота №9.**

**Робота з сокетами**

**Мета роботи:** Вивчити роботу з потоковими сокетами в режимі опиту. Створити сервер котрий буде відповідати на запити клієнтів.

**Короткі теоретичні відомості**

Socket API був вперше реалізований в операційній системі UNIX. Зараз цей програмний інтерфейс доступний практично в будь-якій операційній си-стемі. Хоча всі реалізації чимось відрізняються один від одного, основний набір функцій у них збігається. Спочатку сокети використовувалися в про-грамах на C/C++, але в даний час вони є майже в всіх нових мовах програ-мування (Perl, С#, Java та ін.).

Сокети надають дуже потужний і гнучкий механізм взаємодії між про-цесами (IPC). Вони можуть використовуватися для організації взаємодії про-грам на одному комп'ютері, по локальній мережі або через Інтернет, що доз-воляє вам створювати розподілені додатки різної складності. Крім того, з їх допомогою можна організувати взаємодію з програмами, що працюють під управлінням інших операційних систем.

Сокети підтримують багато стандартних мережевих протоколів (кон-кретний їх список залежить від реалізації) і надають уніфікований інтерфейс для роботи з ними. Найбільш часто сокети використовуються для роботи в IP-мережах.

Сокети, незалежно від виду, поділяються на три типи: потокові, сирі і дейтаграмні. Потокові сокети працюють з установкою з'єднання, забезпечу-ючи надійну ідентифікацію обох сторін і гарантують цілісність і успішність доставки даних, спираючись на протокол TCP. Дейтаграмні сокети працюють без встановлення з'єднання і не забезпечують ні ідентифікації відправника, ні контролю успішності доставки даних, зате вони швидше по-токових, спираючись на протокол UDP. Сирі сокети, вони надають мож-ливість ручного формування TCP \ IP-пакетів.

Також існує 2 види сокетів:

- синхронні – затримують управління на час виконання операції;

- асинхронні – повертають управління, але продовжують виконувати роботу в фоні та після закінчення повідомляють про це.

**Хід роботи**

1. Вихідний код серверу що повертає клієнту число та парність числа зображено в лістингу 1. Вихідний код клієнта що надсилає серверу поточний час зображено в лістингу 2.

Лістинг 1 – Файл серверу

import socket

import threading

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind(("127.0.0.1", 5555))

server.listen(5)

users = []

def send\_message(data, user\_self, sec):

for user in users:

if(sec % 2 == 0):

result = " - Число парне"

data = data.decode("utf-8")

tx = data + result

tx = tx.encode("utf-8")

user.send(tx)

else:

result = " - Число непарне"

data = data.decode("utf-8")

tx = data + result

tx = tx.encode("utf-8")

user.send(tx)

def listen\_user(user):

while True:

try:

data = user.recv(2048)

text = data.decode("utf-8")

sec = text[17:19]

print(sec)

sec\_int = int(sec)

send\_message(data, user, sec\_int)

except:

user.close()

for u in users:

if(u == user):

users.remove(u)

def start():

while True:

socket\_user, adress\_user = server.accept()

print(socket\_user, " is connected")

users.append(socket\_user)

thread\_for\_user = threading.Thread(target=listen\_user, args= (socket\_user,))

thread\_for\_user.start()

if \_\_name\_\_ == '\_\_main\_\_':

start()

Лістинг 2 – Файл клієнта

import socket

import threading

import datetime

import time

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client.connect(("127.0.0.1", 5555))

def listen\_server():

while True:

data = client.recv(2048)

print(data.decode("utf-8"))

def send\_data():

thread\_for\_listen = threading.Thread(target=listen\_server)

thread\_for\_listen.start()

while True:

now = datetime.datetime.now()

client.send(str(now).encode("utf-8"))

time.sleep(1)

if \_\_name\_\_ == '\_\_main\_\_':

send\_data()

Результат виконання програми зображено на рисунку 1. На рисунку видно повернений час та перевірка парності що надійшла клієнту від сервера.

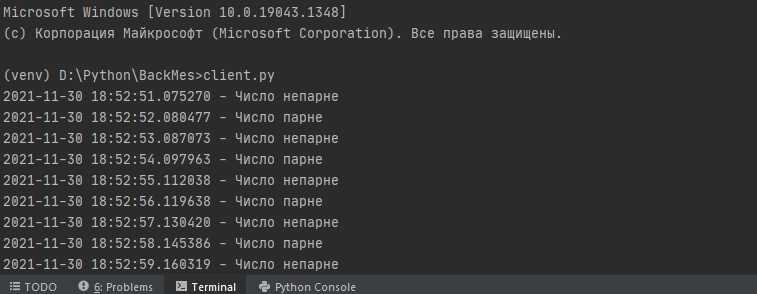


Рисунок 1 - Результат виконання програми що повертає парність секунд

2. Вихідний код сервера та клієнта що працюють на протоколі UDP, зображено в лістингу 3 та лістингу 4 відповідно.

Лістинг 3 – Файл датаграмного серверу

import socket

udp\_server = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

udp\_server.bind(("127.0.0.1", 5555))

def start():

while True:

data, adress\_user = udp\_server.recvfrom(4096)

number = int(data)

for i in range(number):

i = str(i).encode()

udp\_server.sendto(i, adress\_user)

if \_\_name\_\_ == '\_\_main\_\_':

start()

Лістинг 4 – Файл датаграмного клієнта

import socket

import threading

host = "127.0.0.1"

port = 5555

addr = (host, port)

client = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

def listen\_server():

while True:

try:

data, ser = client.recvfrom(4096)

print(data)

except:

pass

def send\_data():

thread\_for\_listen = threading.Thread(target=listen\_server)

thread\_for\_listen.start()

times = input()

if(int(times) > 10 or int(times) < 0):

print("You shoud write in range 0 to 10")

times = input()

client.sendto(str(times).encode("utf-8"), addr)

if \_\_name\_\_ == '\_\_main\_\_':

send\_data()

Результат виконання програми зображено на рисунку 2. Сервер надсилає ту кількість повідомлень яку вказав клієнт, в прикладі вказано – 7.

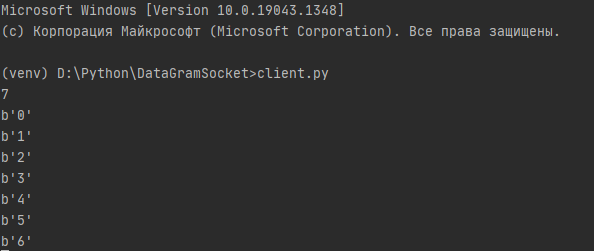


Рисунок 2 – Відповідь клієнту від сервера

3. Вихідний код класу Сокет що використовується для наслідування для класів серверу та клієнта зображено в лістингу 5. Код асинхронного сервера та клієнта зображено в лістингу 6 та 7 відповідно. Сервер приймає зображення для зберігає в директорії проекту в папці з поточною датою.

Лістинг 5 – Файл Сокету

import socket

import asyncio

class Socket:

def \_\_init\_\_(self):

self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.main\_loop = asyncio.new\_event\_loop()

async def send\_data(self, data=None):

raise NotImplementedError()

async def listen\_socket(self, listened\_socket=None):

raise NotImplementedError()

async def main(self):

raise NotImplementedError()

def start(self):

self.main\_loop.run\_until\_complete(self.main())

def set\_up(self):

raise NotImplementedError()

Лістинг 6 – Файл асинхронного серверу

from Socket import Socket

import datetime

import os

class Server(Socket):

def \_\_init\_\_(self):

super(Server, self).\_\_init\_\_()

self.users = []

def set\_up(self):

self.socket.bind(("127.0.0.1", 5555))

self.socket.listen(5)

self.socket.setblocking(False)

print("Server is listening")

async def send\_data(self, data=None):

for user in self.users:

await self.main\_loop.sock\_sendall(user, data)

async def listen\_socket(self, listened\_socket=None):

if not listened\_socket:

return

now = datetime.datetime.now()

str\_date = str(now.year) + "-" + str(now.month) + "-" + str(now.day) + "-" + str(now.hour) + "-" + str(now.minute)

os.mkdir(str\_date)

filetodown = open(str\_date + "/ImgGet.jpg", "wb")

while True:

data = await self.main\_loop.sock\_recv(listened\_socket, 2048)

filetodown.write(data)

async def accept\_sockets(self):

while True:

user\_socket, address = await self.main\_loop.sock\_accept(self.socket)

print(user\_socket)

self.users.append(user\_socket)

self.main\_loop.create\_task(self.listen\_socket(user\_socket))

async def main(self):

await self.main\_loop.create\_task(self.accept\_sockets())

if \_\_name\_\_ == '\_\_main\_\_':

server = Server()

server.set\_up()

server.start()

Лістинг 7 – Файл клієнта

from Socket import Socket

import asyncio

class Client(Socket):

def \_\_init\_\_(self):

super(Client, self).\_\_init\_\_()

self.filetosend = open("img.jpg", "rb")

def set\_up(self):

self.socket.connect(("127.0.0.1", 5555))

self.socket.setblocking(False)

async def listen\_socket(self, listened\_socket=None):

while True:

data = await self.main\_loop.sock\_recv(self.socket, 2048)

print(data.decode("utf-8"))

async def send\_data(self, data=None):

data = self.filetosend.read(128)

while data:

print("Sending")

await self.main\_loop.sock\_sendall(self.socket, data)

data = self.filetosend.read(128)

#await self.main\_loop.sock\_sendall(self.socket, b'Done')

print("Done Sending.")

async def main(self):

await asyncio.gather(

self.main\_loop.create\_task(self.listen\_socket()),

self.main\_loop.create\_task(self.send\_data())

)

if \_\_name\_\_ == '\_\_main\_\_':

client = Client()

client.set\_up()

client.start()

Результат створення папки з назвою поточної дати отримання повідомення зображено на рисунку 3. Вміст папки та перевірка отриманого зображення зображено на рисунку 4.

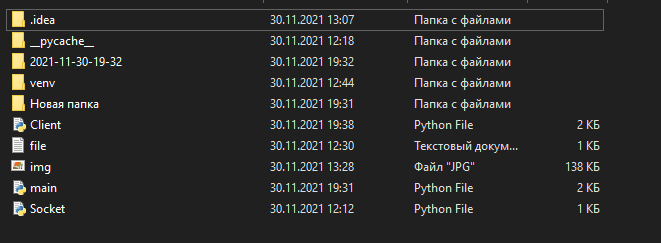


Рисунок 3 – Результат створення папки

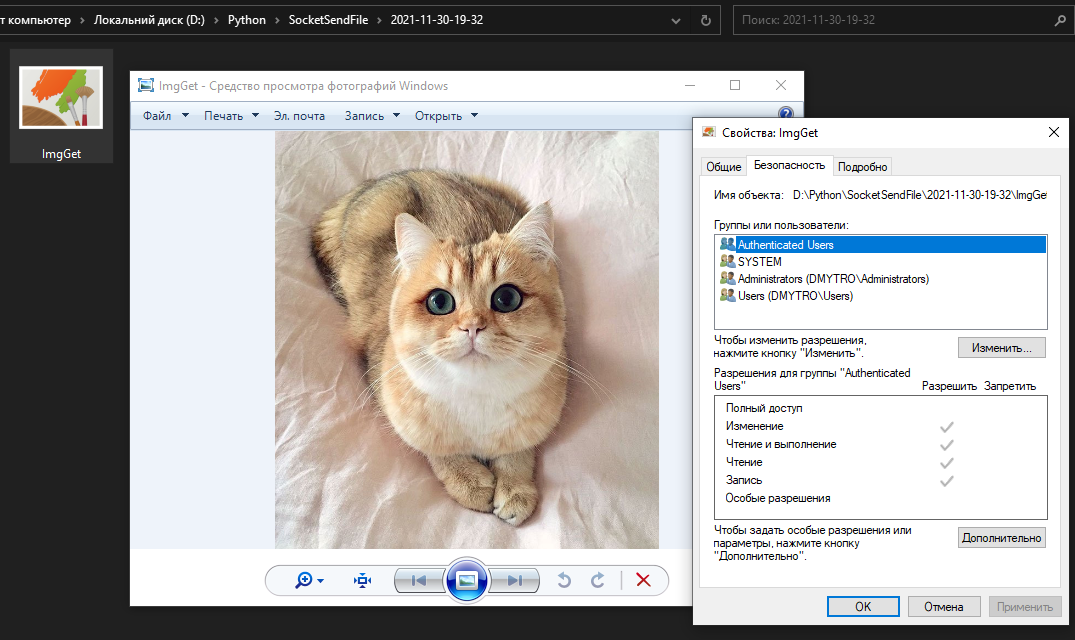


Рисунок 4 – Вміст папки та перевірка отриманого зображення

Висновок: на даній лабораторній роботі я виконав реалізацію потокових, датаграмних та асинхронних в клієнт-серверних застосунках. Виконав відправку зображення в асинхронному додатку.