# ЛАБОРАТОРНА РАБОТА № 6 Дослідження роботи асинхронних сокетів

## Мета роботи: вивчення роботи асинхронних сокетами в режимі потоків виконання, тобто в режимі, коли робота з кожним клієнтом проводиться в окремому серверному потоці.

* 1. Теоретичні відомості

Як ви пам’ятаєте, синхронні сокети затримують управління на час виконання операцію, в свою чергу асинхронні повертають управління, але продовжують роботу в фоні та після закінчення повідомляють про це.

У випадку з синхронними сокетами сервер прийнявши нового клієнта працює з ним (обмінюється інформацією), але інші клієнти чекають в черзі, поки сервер не завершить роботу з цим. Асинхронні сокети працюють паралельно - витягує клієнта з черги, породжує потік/процес, передає йому дескриптор клієнта (котрий повернула функція accept), цей потік/процес починає працювати в фоні, в свою чергу сервер знову витягує нового клієнта з черги і так далі.

Асинхронні сокети слід використовувати там, де є велике навантаження при передачі даних.

Реалізація програмної частини залишається такою ж як і при синхронних сонетах. Винятком є те, що ми оброблюємо інформацію в потоці, а саме, витягнувши запит від клієнта з черги, ми передаємо його дескриптор в функцію, котра виконується в потоці.

**Хід роботи**

Вихідний код класу Сокет що використовується для наслідування для класів серверу та клієнта зображено в лістингу 1. Код асинхронного сервера та клієнта зображено в лістингу 2 та 3 відповідно. Сервер приймає зображення для зберігає в директорії проекту в папці з поточною датою.

Лістинг 1 – 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 2 – 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 + "/photo\_get.png", "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 3 – Файл клієнта

from Socket import Socket  
import asyncio  
  
class Client(Socket):  
 def \_\_init\_\_(self):  
 super(Client, self).\_\_init\_\_()  
 self.filetosend = open("photo.png", "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()

Висновок: на даній лабораторній роботі я вивчив роботу асинхронних сокетів в режимі потоків виконання, тобто в режимі, коли робота з кожним клієнтом проводиться в окремому серверному потоці.