... continued

This lesson discusses the general concept of a coroutine.

We'll define an uber coroutine that will create the server and instantiate the users.

```
async def main():
    server_port = random.randint(10000, 65000)
    server_host = "127.0.0.1"
    chat_server = ChatServer(server_port)
    jane = User("Jane", server_host, server_port)
    zak = User("Zak", server_host, server_port)

    server = await asyncio.start_server(chat_server.run_server
, server_host, server_port)
    asyncio.create_task(jane.run_client())
    asyncio.create_task(zak.run_client())

    await server.serve_forever()
```

Note that we are using the **`asyncio.create_task()`** API to schedule the execution of the **`run_client()`** API for each user object. Finally, we'll use the **`asyncio.run()`** method to run the **`main()`** coroutine. The complete code appears below:

```
class ChatServer:

    def __init__(self, port):
        self.port = port
        self.clients = {}
        self.writers = {}

    async def handle_client(self, message, writer):

        command, param = message.split(",")

        if command == "register":
            print("\n{0} registered -- {1}\n".format(param, cu
rrent_thread().getName()))
            self.clients[param] = writer
            self.writers[writer] = param

            # send ack
            writer.write("ack".encode())
            await writer.drain()

        if command == "chat":
            to_writer = None
            if param in self.clients:
                to_writer = self.clients[param]

            if to_writer is not None:
                to_writer.write(("{0} says hi".format(self.wri
ters[writer])).encode())
                await to_writer.drain()
            else:
                print("\nNo user by the name |{0}|\n".format(p
aram))

        if command == "list":
            names = self.clients.keys()
            names = ",".join(names)
            writer.write(names.encode())
```

```
            await writer.drain()


    async def run_server(self, reader, writer):

        while True:
            data = await reader.read(4096)
            message = data.decode()
            print("\nserver received: {0} -- {1}\n".format(mes
sage, current_thread().getName()))

            await self.handle_client(message, writer)


class User:

    def __init__(self, name, server_host, server_port):
        self.name = name
        self.server_port = server_port
        self.server_host = server_host

    async def receive_messages(self, reader):

        while 1:
            message = (await reader.read(4096)).decode()
            print("\n{0} received: {1} -- {2}\n".format(self.n
ame, message, current_thread().getName()))

    async def run_client(self):
        reader, writer = await asyncio.open_connection(self.se
rver_host, self.server_port)

        # register
        writer.write("register,{0}".format(self.name).encode
())
        await writer.drain()
        await reader.read(4096)

        # get list of friends
        writer.write("list,friends".encode())
        await writer.drain()
        friends = (await reader.read(4096)).decode()
```

```
        friends = (await reader.read(4096)).decode()
        print("Received {0}".format(friends))


        # launch coroutine to receive messages
        asyncio.create_task(self.receive_messages(reader))

        friends = friends.split(",")
        num_friends = len(friends)

        while 1:
            friend = friends[random.randint(0, num_friends - 1
)]
            print("{0} is sending msg to {1} -- {2}".format(se
lf.name, friend, current_thread().getName()))
            writer.write("chat,{0}".format(friend).encode())
            await writer.drain()
            await asyncio.sleep(3)


async def main():
    server_port = random.randint(10000, 65000)
    server_host = "127.0.0.1"
    chat_server = ChatServer(server_port)
    jane = User("Jane", server_host, server_port)
    zak = User("Zak", server_host, server_port)

    server = await asyncio.start_server(chat_server.run_server
, server_host, server_port)
    asyncio.create_task(jane.run_client())
    asyncio.create_task(zak.run_client())

    await server.serve_forever()


if __name__ == "__main__":
    # start server
    asyncio.run(main())
```

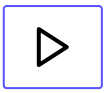
You can run the above code in the widget below, however, the code is tweaked slightly to work with Python-3.5 since the code widget runs that

version of Python.

```
from threading import current_thread
import time, random, asyncio


class ChatServer:

    def __init__(self, port):
        self.port = port
        self.clients = {}
        self.writers = {}

    async def handle_client(self, message, writer):

        command, param = message.split(",")

        if command == "register":
            print("\n{0} registered -- {1}\n".format(param, current_thread().getNam
            self.clients[param] = writer
            self.writers[writer] = param

            # send ack
            writer.write("ack".encode())
            await writer.drain()

        if command == "chat":
            to_writer = None
            if param in self.clients:
                to_writer = self.clients[param]

            if to_writer is not None:
                to_writer.write(("{0} says hi".format(self.writers[writer])).encode
                await to_writer.drain()
            else:
                print("\nNo user by the name |{0}|\n".format(param), flush=True)

        if command == "list":
            names = self.clients.keys()
            names = ",".join(names)
            writer.write(names.encode())
            await writer.drain()

    async def run_server(self, reader, writer):

        while True:
            data = await reader.read(4096)
            message = data.decode()
            print("\nserver received: {0} -- {1}\n".format(message, current_thread(

            await self.handle_client(message, writer)
```

```
class User:


    def __init__(self, name, server_host, server_port):
        self.name = name
        self.server_port = server_port
        self.server_host = server_host

    async def receive_messages(self, reader):

        while 1:
            message = (await reader.read(4096)).decode()
            print("\n{0} received: {1} -- {2}\n".format(self.name, message, current

    async def run_client(self):
        reader, writer = await asyncio.open_connection(self.server_host, self.serve

        # register
        writer.write("register,{0}".format(self.name).encode())
        await writer.drain()
        await reader.read(4096)

        # get list of friends
        writer.write("list,friends".encode())
        await writer.drain()
        friends = (await reader.read(4096)).decode()
        print("Received {0} -- {1}".format(friends, current_thread().getName()), fl

        # launch coroutine to receive messages
        asyncio.ensure_future(self.receive_messages(reader))

        friends = friends.split(",")
        num_friends = len(friends)

        while 1:
            friend = friends[random.randint(0, num_friends - 1)]
            print("{0} is sending msg to {1} -- {2}".format(self.name, friend, curr
            writer.write("chat,{0}".format(friend).encode())
            await writer.drain()
            await asyncio.sleep(3)

            #if self.name == "Jane":
            #    print("Jane about to go to sleep")
            #    time.sleep(1000)


async def main():
    server_port = random.randint(10000, 65000)
    server_host = "127.0.0.1"
    chat_server = ChatServer(server_port)
    jane = User("Jane", server_host, server_port)
    zak = User("Zak", server_host, server_port)

    await asyncio.start_server(chat_server.run_server, server_host, server_port)
    zak_task = asyncio.ensure_future(zak.run_client())
```

```
    zak_task = asyncio.ensure_future(zak.run_client())
    jane_task = asyncio.ensure_future(jane.run_client())

    await zak_task, jane_task


if __name__ == "__main__":

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```



The simulation run in the code widget will timeout as we don't implement a graceful shutdown of the event loop. However, of importance is to note the thread name that is printed in each statement in the output. It is exclusively **MainThread**. The simulation can keep on running forever without blocking using a single thread! This is the magic of asyncio.

Finally, as an exercise, uncomment **lines#92 - 94**, which introduces a synchronous blocking sleep for one of the users, and observe from the output that the entire application stops.

← Back

Next →

... continued

Quiz

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/continued__asyncio__python-concurrency-for-senior-engineering-interviews)