CPython Async Programming

Sean Xu@Bitsoda

soda-quant

a bunch of tasks in a single asyncio eventloop

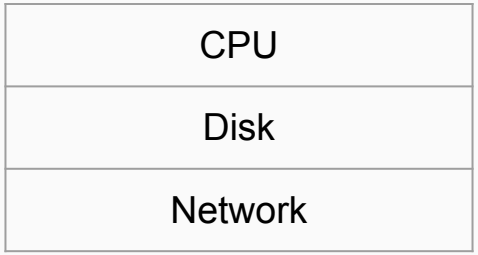
Q1: How do they cooperate

Q2: Why choose asyncio

Why async?

System Event	Actual Latency	Scaled Latency
One CPU Cycle	0.4 ns	1 s
Level 1 cache access	0.9 ns	2 s
Level 2 cache access	2.8 ns	7 s
Level 3 cache access	28 ns	1 min
Main memory access	~ 100 ns	4 min
...		
NVMe SSD I/O	~ 25 µs	17 hrs
SSD I/O	50 - 150 µs	1.5 - 4 days
Rotational disk I/O	1 - 10 ms	1 - 9 months
Internat call	65 ms	5 years

The Answer: I/O is SLOW



Final Solution For the C10K Problem

Process	Apache, Tomcat (Gunicorn) ...
Thread	
Async Programming	Nginx (Tornado) ...

Asyncio in Python

generator

interrupt

resume

yield

generator

coroutine

PEP 342 Coroutines via Enhanced Generators

PEP 380 Syntax for Delegating to a Subgenerator

coroutine

asyncio

Future

Task

Eventloop

In Brief

Can break / restore	generator
Can co-operate	coroutine
Standard Scheduler	asyncio

Implementing A IOLOOP

Practice

Future

Task

IOLoop

simple ioloop

Using Asyncio

Practice

ChatServer

create server
protocol

Q/A

Thanks!

```
import asyncio
from itertools import chain

TELNET_EOL = "\r\n"


class Protocol(asyncio.Protocol):

    def __init__(self, chat_room):
        self._chat_room = chat_room
        self._username = None
        self._transport = None
        self._buffer = []

    def connection_made(self, transport):
        self._transport = transport
        self._writeline(f"Welcome to {self._chat_room.name}")
        self._write("Enter user name: ")

    def connection_lost(self, exc):
        self._deregister_user()

    def data_received(self, raw_data: bytes):
        try:
            data = raw_data.decode("utf-8")
        except UnicodeDecodeError as e:
            self._transport._write(str(e).encode("utf-8"))
        else:
            for line in self._accumulated_limes(data):
                self._handle(line)

    def _accumulated_limes(self, data):
        self._buffer.append(data)
        while True:
            tail, newline, head = self._buffer[-1].partition(TELNET_EOL)
            if not newline:
                break
            line = "".join(chain(self._buffer[:-1], (tail,)))
            self._buffer = [head]
            yield line

    def _handle(self, line):
        if self._username is None:
            self._register_user(line)
        elif line.upper() == "NAMES":
            self._list_users()
        else:
            self._chat_room.message_from(self._username, line)

    def _register_user(self, line):
        username = line.strip()
        if self._chat_room.register_user(username, self._transport):
            self._username = username
        else:
            self._writeline(f"Username {username} not available")

    def _deregister_user(self):
```

```
        if self._username is not None:
            self._chat_room.deregister_user(self._username)

    def _list_users(self):
        self._writeline("Users here:")
        for username in self._chat_room.users():
            self._write("   ")
            self._writeline(username)

    def _writeline(self, line):
        self._write(line)
        self._write(TELNET_EOL)

    def _write(self, text):
        self._transport.write(text.encode("utf-8"))

# ########################

import asyncio
import sys

from chat_room.chat_server_protocol import Protocol, TELNET_EOL


class ChatRoom(object):

    def __init__(self, name, port, loop):
        self._name = name
        self._port = port
        self._loop = loop
        self._username_transports = {}

    @property
    def name(self):
        return self._name

    def run(self):
        coro = self._loop.create_server(  # create_server returns a tcp server object
            protocol_factory=lambda: Protocol(self),
            host="",
            port=self._port
        )
        return self._loop.run_until_complete(coro)

    def register_user(self, username, transport):

        if username in self.users():
            return False
        self._username_transports[username] = transport
        self._broadcast(f"User {username} arrived.")
        return True

    def deregister_user(self, username):
        del self._username_transports[username]
        self._broadcast(f"User {username} departed.")

    def users(self):
        return self._username_transports.keys()

    def message_from(self, username, message):
```

```
118         self._broadcast(f"{username}: {message}")
119
120     def _broadcast(self, message):
121         for transport in self._username_transports.values():
122             transport.write(message.encode("utf-8"))
123             transport.write(TELNET_EOL.encode("utf-8"))
124
125
126 def main():
127     name = sys.argv[1] if len(sys.argv) >= 2 else "ChatServer"
128     port = sys.argv[2] if len(sys.argv) >= 3 else 1234
129
130     print(f"{name} running on port {port}")
131
132     loop = asyncio.get_event_loop()
133     chat_room = ChatRoom(name, port, loop)
134     _server = chat_room.run()
135     loop.run_forever()
136
137
138 if __name__ == '__main__':
139     main()
```

PDF document made with CodePrint using Prism