Asynchronous I/O with server herd

Chen Chen

I. Abstract

This report descibes my implementation on server which uses Asynchronous I/O as the method to Input and output. Also this report analyses on python's asyncio library's framework and descibes the pros and cons of using python's asyncio library. In the end this report compares python with Node.js.

II. implementation

A. Structure

The server initiates with three Dictionary types value. The Server_Connect value which uses name of Mainframe as key and return a list of name of other server stores the name of other servers which can be connected by Mainframe. The Server_Value which uses name of the server as key and return a tuple of ip address and port of server stores the ip address and the port of Mainframe and other server. The clients_dict which uses clients's name as key and return a tuple of clients's information stores information about clients. The server contains two different class Object. The EchoServer is the class object for receiving data from other servers or clients and in charge with commucating with Google Place API Web Server. The EchoClient is the class object for propagating place information to other servers.
The server contains one Log Object. The Server_Log using python's logging library will create a log file when the server is initiated and marks server error and debug information. The Level of Server_Log is INFO.
The server contains a event loop object. The event_loop using python's asyncio library implements the asynchronous I/O

B. Class' method

1) EchoClient: EchoClient handles communication with server.

```
__ini__(self, message):
```

The EchoClient constructor accepts one arguments. A string of message to send. This method triggers Server_Log to mark a INFO level event.

```
connection_made(self, transport):
```

When the client successfully connects to the server, it triggers connection_made method. It starts communicate with server immediately, but actually the transport object buffers the outgoing data until the socket's buffer is ready to receive data. This is all handled transparently, so the application code can be written as though the I/O operation is happening right away. This method triggers Server_Log to mark down all message is sent as INFO level event.

```
data_received(self, transport):
```

When the response from the server side, it will triggers data_received. This method triggers Server_Log to mark down the response from server as a INFO level event.

```
eof_received(self):
connnection_lost(self, exc):
```

The connection close from the server side triggers connnection_lost.
The End-of-file marker received triggers eof_received.
Both of methods closes the local transport and triggers Server_Log to mark down a INFO level event.

2) EchoServer: :EchoServer handles client communication and the commucation with Google API Web Server.

```
__init__(self, name):
```

The EchoServer constructor accepts one arguments. A string of message to send. This method triggers Server_Log to mark a INFO level event.

```
connection_made(self, transport):
```

Each new clients connection with listening port triggers connection_made. The instance of asyncio.Transport object is provided in argument which provides an asynchronous I/O abstraction. The asyncio.Transport implements different types of commucation with same API and provides get_extra_info to achieve client's IP address and port. This method triggers Server_Log to mark down a INFO level event.

```
data_received(self, data):
```

Once the connection is established, data is sent by client, which triggers data_received. The data is pass as byte string. After decoding the data, it calls procString method to deal with message. This method triggers Server_Log to mark down a INFO level event.

```
connection_lost(self, error):
```

Once the transport is closed connection_lost is triggered. If there is a error appear because of connection_lost argument error will be set as true. This method could trigger Server_Log to mark down ERROR level event if error appear, or INFO level event if nothing happen.

```
check_coords(self, coord_str):
```

check the legality of a coordination string. This method could raise a ValueError if coordination the string is illegal, and could trigger Server_Log to mark down a ERROR level error if error appear.

```
coord_to_tuple(self, coord_str, msg):
```

This method spilts a coordination string into two part, and return a tuple contains latitude string in zero position and longitude in the first position.

```
get_html(self, url):
```

A coroutine with url link as argument return the response html file of url link in argument. This coroutine triggers Server_Log to mark down a INFO level event.

```
handler_client(self, server_name, task):
```

After server completes sending message to other servers, this method will be called by task object by task's method add_done_callback. This method can received the return value from the task argument by calling task's result method. This method implements binding callback. This method could raise any types of exceptation, and the Server_Log can mark down a ERROR event when error appears.

```
def handler_html(self, items, task):
```

After server completes commucation with Google API Web Server, this method will be called by task object by task's method add_done_callback. This method can received the return value from the task argument by calling task's result method and modifies the json file as requirement and send the json file to client and could triggers any types of exceptation and triggers Server_Log to mark down a ERROR level event if error appear or a INFO level if nothing happen and closes the local transport port.

```
errorhandler(self, msg):
```

A error handler. This method trigger Server_Log to mark down a ERROR level event, and send the message back to client with a '?' in front of message and closes the local transport.

```
procString(self, msg):
```

This method confirms the command from the message and calls proc_IAMAT or proc_AT or proc_WHATSAT depend on the message and could raise a error if the command is not vaild. If error appear, this method will call errorhandler to proccess message.

```
proc_IAMAT(self, msg):
```

Proccess handler for command IAMAT. This method could call errorhandler if message is a invaild IAMAT command and form a AT command and send back AT command to client side and call proc_AT method to proccess AT command and triggers the Server_Log to mark down a INFO level event.

```
proc_AT(self, msg):
```

Proccess handler for command AT. This method could call errorhandler if message is a invaild AT command. This method could put client into current data base if client doesn't exist or is a newer version and propagate the AT command with other server by using EchoClient class and put it into event_loop object to achieve asynchronous I/O. This method triggers Server_Log to mark down a INFO level event and two possible INFO level event when saving data to data base and one ERROR level event if there is error appear when creating coroutine and putting coroutine into event loop.

```
proc_WHATSAT(self, msg):
```

Procces handler for command WHATSAT. This method could call errorhandler if meesage is a invaild WHATSAT command. This method creates coroutine and puts it into event loop. This method triggers Server_Log to mark down a INFO level event.

3) Notes: Both class is the subclass of asyncio.Protocol, and protocol object's methods are invoked based on events associated with the server socket. As a result, the server is built with event-driven concurrency principle. The event loop provided by asyncio library works only when clients connected to the port. On other time, the event loop need extreme low operation system resource and hard ware resource to listen to the port.

III. Conclusion

asyncio

pros:

1. It is very good for asynchronous I/O tasks, because this library was designed for asynchronous I/O. Compared to the gevent library in python2, the efficiency of asyncio library is much higher.
2. Programming model is simpler, the burden is relatively small. Because by using asyncio object to get an event loop and then just need to plug coroutine into this loop.
3. The operationg system still view such program as signle - threaded, but at a macroscopic level it sees multiple concurrency.
4. The pros of asynchronous I/O is it doesn't block the program execution while reading or sending data. Compared to Blocking I/O and Non-Blocking I/O and I/O multiplexing which block while reading or sending message, asynchronous I/O improve the efficiency a lot. It like hiring somebody else help you fishing and cooking, and only going to notice you when the fish is ready to eat.

cons: 1. It doesn't allow differnt I/O model in server. If a application decided to use asyncio library, it must assure the from front-end to back-end all use the asynchronous I/O.

Python

Dynamic type checking allows server to quickly proccess command string and json file. However, also because of dynamic type checking, it is extreme hard for programming check type error or syntax error in compile time. If such bug haven't been found during writing code, and update server by dynamic linking, it will cause catastrophe.

Python uses reference counting as memory management. It work prefectly in server port, because memory management malloc RAM resource to objects while connection is made, when nothing point to them the counter became 0, then memory management free the RAM which the object occupied. It work prefectly well on server port, when nothing point to object mean the abstraction connection of object is close, on server means never use it again, free such memory is the prefectly choose. However it need large RAM resource to store counter information, compared to other language it lower the efficiency and could cause memory leaking, if the counter couldn't successfully determind which object should be free.

Syntax issue with python: python doesn't use parantheses and semicolon is a catastrophe in my opinion. Because in my opinion semicolon and parantheses help programmer format the code, so code is much easier for people to read. Though I don't need to consider management of the project, It is terrible for people to debug when deal with large python-based project.

A. *Comparsion with Node.js*

Node.js is much faster than python. Becuase first, Node.js is using Chrome V8 is extreme fast even faster than java virtual machine. Second, asynchronous call. The function of Node.js is based on the V8 chrome's asynchronous network and IO library, though it is similar with asyncio library, but also because V8 it is much faster.

Python is easier to use and write. Also because python's attributes lots of library and package can be use in python-based project. Python provides with good scalability and programmabiliry.