Google Summer of Code 2021

Boost.Python: asyncio Eventloop running in strand

Pan Yue

April 9, 2021

Contents

1 Personal Details

Name Pan Yue

Email philoinovsky@gmail.com

Major Electrical and Electronic Engineering, Year 3, B.Eng

University Nanyang Technological University, Singapore

GitHub @philoinovsky

Timezone UTC+8

Availability ~20 hours/week, June 8 - August 17, no other factors will affect

2 Background Information

2.1 Educational Background

Learning in School

I am a third year engineering undergraduate from School of Electrical and Electronics Engineering in Nanyang Technological University, Singapore. The school provides a holistic academic curriculum in 3 aspects namely Computing, Electrical Engineering and Electronic Engineering with broader perspectives such that I am well balance with knowledge skill and ability in hardware (e.g. digital electronic, signals and systems) and software through hands-on and practical experiential learning. In addition, the school enable me to solve problem critically with analytical and logical thinking

I achieved all Distinction Grade in computing modules such as Data Structures and Algorithms, Introduction to Artificial Intelligence, Microprocessor. By accomplishing these modules, it ascertains my career interest towards software engineer due to my passion in software coding and programming.

Learning Online

Due to my IT passion, I am motivated to initiate by studying more courses through virtual training and learning development provided by MIT; those courses (e.g. Operating System, Distributed System, Computer Language Engineering [a.k.a Principles of Compiler Design]) involved intensive software coding projects.

2.2 Programming Background

Internships

- ByteDance Ltd, **Software Engineering Intern** May 2021 – Aug 2021
- Shopee Pte Ltd, **Software Engineering Intern** Jan 2021 – May 2021

  - Project 1: use `Python` to implement git log formatting and parsing tools. Then serve those files through `Django`
  - Project 2: use Java to build internal services.

- Nanyang Technological University, **Research Assistant** Apr 2020 – Jul 2020
  - Project 1: use `Django` implementing services for user locating and routing.
  - Project 2: use `Python` scripts on Raspberry Pi to utilize Bluetooth for three-point locating.
  - Project 3: use `React Native JS` to judge user mobile phone velocity/acceleration and predict the future position.

Programming Interests

I had applied C++ programming in competitive programming and school projects; I realized that C++ is a robust programming language that could create positive impacts in daily life, commercial and engineering industries in terms of contributions. Hence, I wish to join the potential workforce in Boost C++ for contribution and learning from their experts and mentors. By executing project involving Boost.Python and Boost.Asio, it will hone my software knowledge, skill and ability with insight onto about C++ deployment and applications.

Why Boost.Python

This is **the only proposal** I submitted because of my strong interest in this topic.
Python has rich ecosystem. This project will benefit those who want to use strand-based event-loop in C++. By combining Python and C++ could lead more efficient accessibility thereby achieving operations productivity.
There are existing solutions like boost.Python, cpython and swig. I am excited to know Boost C++ had project for students on GSoC providing opportunities to learn and contribute with C++/Python API. For me, it is a great entry point to start OSS journey.

Future Plans

Beyond this summer project, I will have insights into Boost.Python and Boost.Asio. Upon accomplishing assigned projects for implementation. I could value-add by enhancing more potential features, solving and rectifying technical bugs, developing more test cases to these two libraries.

Rate Skills (0 to 5)

C++ 98/03 (traditional C++)	4.5	through projects accomplishments
C++ 11/14 (modern C++)	4.5	learnt from book
C++ Standard Library	4.5	familiar
Boost C++ Libraries	4	familiar with Boost.Python and Boost.Asio, but need to refer to docs and read source code often
Git	4.5	day-to-day use

Software Development Environment

OS: `Windows Subsystem for Linux 2 (WSL2)`, Macbook

IDE: `VSCode` (it can connect to host via ssh), `Intellij` (local Java development)

Software Documentation Tool

Haven't use any. During internship, we document on confluence, which is similar to writing blogs.

3 Project Proposal

3.1 Overview

In some scenarios, GIL could be the performance bottleneck of Python code. There have been a long time that the Python developers are struggling to solve the performance issue brought by `GIL`. In PEP554, people came up with the solution of running multiple interpreters concurrently. But due to some reasons, it wasn't be implemented in Python 3.9.

Thanks to boost::asio::io_context::strand, it's possible for us to use multiple Python interpreters in the same program in the `actor model` fashion. Hereby I propose this project to implement Python eventloop in such a way that it runs in `boost::asio::io_context::strand`

Those developers who wants to use numerous Python libraries and rich Python ecosystem will benefit from this project.

The task is described in GSoC 2021 idea list.

Deliverables

- implementation of Abstract Eventloop running in Boost.Asio strand
- implementation of Policy Objects
- test cases
- documentations

3.2 Event Loop Methods

Refer to this page for the methods need to be implemented.

call_soon and call_when_something_completed

The most fundamental function in this class is `call_soon` and `call_when_something_completed`. Sections Scheduling callbacks, Scheduling delayed callbacks, Watching file descriptors and Working with socket objects directly will be implemented using this mechanism.

- call_soon can be realized by io_service::strand::post as default (as in programming competency task).
- `call_when_something_completed` is more complicated, basically it calls the callback function once the IO execution (network connections, file descriptors, signal handler, pipes, sockets) or the timer completed.
  - For IO execution, it can be realized by using boost::asio::posix::descriptor::async_wait.
  - For timer (i.e function loop.call_later() and loop.call_at()), boost::asio::basic_waitable_timer::async_wait may be useful in the implementation.

Network

This part involves three sections: Opening Network Connections, TLS Upgrade and DNS

- Opening Network Connections: there are three functions under this section: `create_connection`, `create_datagram_endpoint` and `create_unix_connection` which have something in common.
  - they return two objects: `transport` and `protocol`
  - the `protocol` can be created by calling parameter `protocol_factory`
  - for `transport`, can refer to the Python SelectorTransport implementation. The C++ version will be using bsd_socket, and methods of `protocol` will be called asynchronously inside eventloop during corresponding operations in `transport`
  - I will use `boost::asio::ip::tcp` to implement `create_connection`, use `boost::asio::ip::udp` to implement `create_datagram_endpoint` and use boost::asio::local::stream_protocol to implement `create_unix_connection`
  - this Python create_connection implementation is for reference
- TLS Upgrade includes one function `start_tls` to upgrade `transport` to a new one with `ssl`. Opening Network Connections. boost::asio::ssl library will be used.
- DNS consists of an asynchronous version of `getaddrinfo` and `getnameinfo`, which can be implemented by `post`-ing the c library getaddrinfo/getnameinfo to `strand`.

Error Handling API

For Error Handling API. the proposed solution is simple:

- `handler` will be stored in a variable when user calls `loop.set_exception_handler(handler)`
- the stored handler will be executed when user calls `loop.call_exception_handler(context)`

Summary

s To sum up, I propose following implementations for methods defined in Event Loop Methods. During the GSoC timeframe, I don't think there is enough time to implement everything and I value quality more than rushed work, so I will work on these selected sections below. The rest can be leaved for future work.

Methods	Proposed Implementation
Running and stopping the loop	the loop will execute once Python code transfers thread control back to the host
Scheduling callbacks	`io_service::strand::post`
Scheduling delayed callbacks	`io_service::strand::post`, `basic_waitable_timer::async_wait`
Creating Futures and Tasks	-
Opening network connections	`ip::tcp`, `ip::udp`, `local::stream_protocol`
Creating network servers	-
Transferring files	-
TLS Upgrade	`boost::asio::ssl`
Watching file descriptors	`basic_socket::async_wait`
Working with socket objects directly	`basic_socket::async_wait`, `basic_stream_socket::read_some`, `basic_stream_socket::write_some`
DNS	`post getaddrinfo` and `getnameinfo` to `strand`
Working with pipes	-
Unix signals	-
Executing code in thread or process pools	-
Error Handling API	use a variable to store the handler and call it when required
Enabling debug mode	-
Running Subprocesses	-

3.3 Policies

A Policy Objects will be implemented to use the C++ version of `eventloop` as default. The `init` function will be called before executing Python codes in C++

The Idea

Since `Boost C++` is usually used to expose C++ code to Python, not the other way around, I will utilize `Python C API` to subclass from the Python `AbstractEventLoopPolicy` by simulating `childClass = type("childClassName", baseClass, methodTable)`. The child class will return C++ implemented Eventloop when `get_event_loop` is called. After that, the C++ will set the policy as default. The proposed implementation will be similar with the code below.

Code

```
static PyObject *
MyPolicy_get_event_loop(PyObject *self, PyObject *unused)
{
    // return the cpp-implemented EventLoop
}

void init()
{
    // import asyncio
    PyObject *asyncio;
    asyncio = PyImport_ImportModuleNoBlock("asyncio");
    if (asyncio == NULL)
        return NULL;

    // defaultPolicy = asyncio.AbstractEventLoopPolicy
    PyObject* defaultPolicy = PyObject_GetAttrString(asyncio, "
    AbstractEventLoopPolicy");
    if (defaultPolicy == NULL)
        return NULL;

    // ClassMyPolicy = type("MyEventLoopPolicy", defaultPolicy, {
    //         "get_event_loop": MyPolicy_get_event_loop
    // })
    PyObject *name = PyUnicode_FromString("MyEventLoopPolicy");
    if (name == NULL)
        return NULL;

    PyObject *base = defaultPolicy;
    if (base == NULL)
        return NULL;

    PyObject *methodDict = PyDict_New();
    if (methodDict == NULL)
        return NULL;
    if (PyDict_SetItemString(methodDict, "get_event_loop",
    MyPolicy_get_event_loop))
        return NULL;

    PyObject *ClassMyPolicy = PyObject_CallObject(
        (PyObject*)&PyType_Type, PyTuple_Pack(3, name, base, methodDict)
    );
    if (ClassMyPolicy == NULL)
        return NULL;

    // asyncio.set_event_loop_policy(ClassMyPolicy())
    PyObject *ClassMyPolicyInstance = PyObject_Call(ClassMyPolicy, PyTuple_Pack
    (0), NULL);
    if (ClassMyPolicy == NULL)
        return NULL;
```

```
47      PyObject *setEventLoopPolicy = PyObject_GetAttrString(asyncio, "
        set_event_loop_policy");
48      if (setEventLoopPolicy == NULL)
49          return NULL;
50      PyObject_Call(setEventLoopPolicy, ClassMyPolicyInstance, NULL);
51  }
```

Listing 1: Policy Object Example

4 Proposed Timeline

The whole project will be divided into 10 weeks. The last week of each coding phases (Week 5, Week 10) are reserved as a "buffer week" dedicated to fnishing up jobs, testing and fxing bugs. At the end of buffer weeks, the code is expected to be merge-ready.
For each week, I aim to fnish the proposed goals before weekend and then invite my mentor to review the changes. If the project progresses faster than planned, I will spend time on other aspects of this project

The Prologue

- explore into the details of the implementations, try out the possibilities and see what's the best.
- start early

Week 1

Get all `call_*` stuff done, which involves:

- Scheduling callbacks
- Scheduling delayed callbacks

Week 2

Complete all methods related to `async_*`, that is:

- Watching file descriptors
- Working with socket objects directly

Week 3

Misc, implement below sections

- `TLS Upgrade`
- `DNS`
- `Error Handling API`

Week 4
Implement `Opening network connections`

Week 5 (buffer week)
For the 1st Evaluation: implement all the proposed implementations for the `Abstract Event Loop`, make it merge ready.

Week 6
Expose `Eventloop` to Python, which means it can be accessed as a `PyObject` object.

Week 7
Implement `Policy Object` using `Python C API`

Week 8
Test cases, aiming at both correctness and security.

- all callbacks can be executed correctly
- adding callbacks must be thread-safe
- follow the RAII rule for resource management

Week 9
Documentations for both `Eventloop` and `Policy Objects`

Week 10 (buffer week)
For the 2nd Evaluation: make this project merge ready.

5 Programming Competency

Finished under guidance of mentor **Vinícius dos Santos Oliveira**s