

GSoC 2021 Draft Proposal v2 - Boost.Python asyncio Eventloop

Pan Yue

April 2, 2021

Contents

1	Personal Details	2
2	Background Information	2
2.1	Educational Background	2
2.2	Programming Background	2
3	Project Proposal	4
3.1	Overview	4
3.1.1	Deliverables	4
3.2	Event Loop Methods	4
3.2.1	call_soon and call_when_something_completed	4
3.2.2	Network	5
3.2.3	Error Handling API	5
3.2.4	Summary	5
3.3	Policies	6
4	Proposed Timeline	6
5	Programming Competency	8

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](https://github.com/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 I choose Boost.Python

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

Usually, I set up WSL2 of ubuntu on my windows 10 machine. And connect it in VSCode. for c++, golang, python, VSCode is used most often. for Java, I use IntelliJ. But if it's on remote machine, I will connect via ssh in VSCode.

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

[GIL](#) is restricting Python code performance. 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 idea of multi-interpreter in the same program. Which can start multiple interpreter concurrently to solve the problem. 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 using `boost::asio::io_context::strand`

Those developers who wants to use numerous Python libraries and share 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 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 call 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](#). `Socket` will be useful for our implementation, and methods of `protocol` will be called asynchronously inside eventloop during corresponding operations in `transport`
 - [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](#). `OpenSSL` library would be useful for this part.
- [DNS](#) contains just 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 some where when user calls `loop.set_exception_handler(handler)`
- the stored handler will be executed when user calls `loop.call_exception_handler(context)`.

Summary

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	callbacks will be executed after Python code exit
Scheduling callbacks	<code>io_service::strand::post</code>
Scheduling delayed callbacks	<code>io_service::strand::post</code> , <code>basic_waitable_timer::async_wait</code>
Creating Futures and Tasks	-
Opening network connections	<code>sys/socket.h</code>
Creating network servers	-
Transferring files	-
TLS Upgrade	OpenSSL
Watching file descriptors	<code>basic_socket::async_wait</code>
Working with socket objects directly	<code>basic_socket::async_wait</code> , <code>basic_stream_socket::read_some</code> , <code>basic_stream_socket::write_some</code>
DNS	<code>post getaddrinfo</code> and <code>getnameinfo</code> to <code>strand</code>
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 implementation is simple, C++ should build a `BOOST_PYTHON_MODULE` and expose `Eventloop` class to Python, on Python side, it will override the method `get_event_loop`, something like this:

```

1 import cppEventLoop
2
3 class MyEventLoopPolicy(asyncio.DefaultEventLoopPolicy):
4     def get_event_loop(self):
5         return cppEventLoop.get_event_loop()
6
7 asyncio.set_event_loop_policy(MyEventLoopPolicy())

```

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 finishing up jobs, testing and fixing bugs. At the end of buffer weeks, the code is expected to be merge-ready.

For each week, I aim to finish 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

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: make the Abstract Event Loop merge ready

Week 6

Expose Eventloop to Python and Implement Policy Object

Week 7

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 8

Documentations for both Eventloop and Policy Objects

Week 9

research on multi-interpreter for Python. If possible, implement a prototype for this.

Week 10 (buffer week)

For the 2nd Evaluation: make this project merge ready.

5 Programming Competency

Finished under guidance of mentor **Vinícius dos Santos Oliveiras**