asyncio pitfalls

Marius Hegele

ChargeHere GmbH

20 June 2025

we want to run things concurrenctly

```python
async def failing_coro():
    print("running failing coro ...")
    raise Exception("some exception")

async def good_coro():
    print("runnning good coro ...")
    await asyncio.sleep(0.1)
    print("good coro finished")
```

pitfall: not picking up exceptions in tasks

```python
async def main1():
    _ = asyncio.create_task(failing_coro())
    await good_coro()
    print("main1 finished")
```

swallows the exception until the whole program terminates

```
runnning good coro ...
running failing coro ...
good coro finished
main1 finished
Task exception was never retrieved
...
Exception: some exception
```

propagating exceptions

use asyncio.gather or await task

```
async def main2():
    await asyncio.gather(
      failing_coro(), good_coro())
    print("main2 finished")

async def main3():
    failing_task = asyncio.create_task(failing_coro())
    await good_coro()
    await failing_task
    print("main3 finished")
```

both propagate the exception

```
running failing coro ...
runnning good coro ...
Traceback (most recent call last):
...
Exception: some exception
```

problem can be hidden inside a class

```python
class MyTask:
    def start(self) -> None:
        self._task = asyncio.create_task(
          self._main())

    async def _main(self) -> None:
        raise Exception("some exception")

    def shutdown(self) -> None:
        self._task.cancel()


async def main4():
    task = MyTask()
    task.start()
    await asyncio.sleep(0.1)
    print("main4 finished")
```

```
main4 finished
Task exception was never retrieved
...
Exception: some exception
```

use ocppproxy.InterruptibleTask instead: implementation

```python
class InterruptibleTask(BlockingTask):
    ...

    async def blocking_start(self) -> None:
        log.info(f"Starting {self.name} task...")
        self._task = asyncio.create_task(self._main())
        try:
            await self._task
        except asyncio.CancelledError:
            pass

    async def shutdown(self) -> None:
         self._task.cancel()

    async def _main(self) -> None: ...
```

use ocppproxy.InterruptibleTask instead: usage

```python
class InternalLoadOptimizer:

    async def blocking_start(self) -> None:
        self._load_optimizer_loop = InterruptibleTask(
            coroutine=self.load_optimizer_loop(GET_DATA_FREQUENCY_IN_SEC),
            name="InternalLoadOptimizerLoop",
        )
        ...
        await self._load_optimizer_loop.blocking_start()

    async def shutdown(self) -> None:
        if self._load_optimizer_loop is not None:
            await self._load_optimizer_loop.shutdown()
```

or implement BlockingTask interface

```
class ModbusTCPServer(BlockingTask):
    async def blocking_start(self):
        self._server = ModbusTcpServer(
            context=self.context,
            identity=self.identity,
            address=(str(self._config.host), self._config.port),
        )
        await self._server.serve_forever()
```

TaskSetManager managing coroutines dynamically

use cases: capacity group tasks, multiple backend clients

```
class TaskSetManager(BlockingTask):
    async def blocking_start(self, **kwargs) -> None: ...
    async def shutdown(self) -> None: ...
    async def update_coroutines_deferring_start(
      self, tasks: Sequence[BlockingTask]) -> None: ...

class LoadControl:
    capacity_groups: CapacityGroupSet
    capacity_group_task_set: TaskSetManager

    async def update_config(self, config: LoadControlConfig) -> None:
      ...
      await self.capacity_group_task_set.update_coroutines_deferring_start(
          tasks=list(self.capacity_groups.values())
      )
```

–

wait_until_first_completed - spot the error

```python
async def wait_until_first_completed(coroutines: Sequence[asyncio.Task]) -> None:
    _, pending = await asyncio.wait(tasks, return_when=asyncio.FIRST_COMPLETED)
    for task in pending:
        task.cancel()
```

```
running failing coro ...
runnning good coro ...
Task exception was never retrieved
...
Exception: some exception
main5 finished
```

wait_until_first_completed - pick up exceptions

```
async def wait_until_first_completed(coroutines: Sequence[asyncio.Task]) -> None:
    done, pending = await asyncio.wait(tasks, return_when=asyncio.FIRST_COMPLETED, timeout=

    for future in pending:
        future.cancel()
        try:
          await future # pick up ignored exception
        except (asyncio.CancelledError, concurrent.futures.CancelledError):
          pass

    for future in done:
        await future # pick up exception and propagate
```

Themes, fonts, etc

- I use default **pandoc** themes.
- This presentation is made with **Frankfurt** theme and **beaver** color theme.
- I like **professionalfonts** font scheme.

Links

- Matrix of beamer themes: https://hartwork.org/beamer-theme-matrix/
- Font themes: http://www.deic.uab.es/~iblanes/beamergallery/indexby_font.html
- Nerd Fonts: https://nerdfonts.com

Text formatting

Normal text. Italic text and **bold text.** ~~Strike out~~ is supported.

Notes

> This is a note.
>
> > Nested notes are not supported. And it continues.

Blocks

This is a block A

- Line A
- Line B

New block without header.

This is a block B

- Line C
- Line D

Listings

Listings out of the block.

```
#!/bin/bash
echo "Hello world!"
echo "line"
```

Listings in the block

```
#!/bin/bash
echo "Hello world!"
echo "line"
```

Table

Item	Description	Q-ty
Item A	Item A description	2
Item B	Item B description	5
Item C	N/A	100

Single picture

This is how we insert picture. Caption is produced automatically from the alt text.

```
![Aleph 0](img/aleph0.png)
```

Figure 1: Aleph 0

Two or more pictures in a raw

Here are two pictures in the raw. We can also change two pictures size (height or width).

```
![](img/aleph0.png){height=10%}\ ![](img/aleph0.png){height=30%}
```

Lists

1. Idea 1
2. Idea 2

- genius idea A
- more genius 2

1. Conclusion

Two columns of equal width

Left column text.
Another text line.

- Item 1.
- Item 2.
- Item 3.

Two columns of with 40:60 split

Left column text.
Another text line.

- Item 1.
- Item 2.
- Item 3.

Three columns with equal split

Left column text.
Another text line.

Middle column list:

1. Item 1.
2. Item 2.

Right column

- Item 1.
- Item 2.

Three columns with 30:40:30 split

Left column text.
Another text line.

Middle column list:

1. Item 1.
2. Item 2.

Right column list:

- Item 1.
- Item 2.

Two columns: image and text

Text in the right column.
List from the right column:

- Item 1.
- Item 2.

Two columns: image and table

Item	Option
Item 1	Option 1
Item 2	Option 2

Fancy layout

Proposal

- Point A
- Point B

Pros

- Good
- Better
- Best

Cons

- Bad
- Worse
- Worst

Conclusion

- Let's go for it!
- No way we go for it!