microasync Documentation

Release 0.1

Vladimir Iakovlev

October 23, 2014

Contents

Green threads and CSP for micropython.

Api documentation.

CHAPTER 1

Installation

For installing run:

```
pip-micropython install microasync
```

CHAPTER 2

Basic usage

For basic usage you should create coroutines and start main loop. For example, script that prints *ok!* every ten seconds:

```
from microasync.async import loop, coroutine, Delay


@coroutine
def main_coroutine():
    while True:
        print('ok!')
        yield Delay(10)


main_coroutine()
loop()
```

More examples:

- examples in repo;
- green threads on pyboard;
- csp on pyboard;
- frp on pyboard.

Api documentation:

2.1 microasync package

2.1.1 Submodules

2.1.2 microasync.async module

class microasync.async.**Channel**(*limit=1*)

Bases: microasync.utils.WithEquality

Channel for communicating between coroutines.

Usage (should be used only inside coroutine):

```
chan = Channel()
yield chan.put('test')  # puts 'tests' in channel
result = yield chan.get()  # gets value from channel
print(result)  # prints 'test'
```

get()
Get item from channel.

Returns Promise for getting item from channel.

Return type Promise

process()
Process promises in queue. For internal use only!

put(*val*)
Put item in channel.

Returns Promise for putting item in channel.

Return type Promise

class microasync.async.**ChannelProducer**(*chan*)
Bases: object

Helper for creating channel clones.

Usage:

```
producer = ChannelProducer(chan)
chan_1 = producer.get_clone()
chan_2 = producer.get_clone()
```

get_clone()
Creates a clone of original channel.

Returns Created channel.

Return type Channel

class microasync.async.**CoroutineBlock**(*gen*)
Bases: microasync.utils.Promise

Similar to go-block in core.async. Internal use only!

parked
Returns *True* when block parked.

process()
Process promises returned by coroutine generator.

class microasync.async.**Delay**(*sec*)
Bases: microasync.utils.Promise

Channel/Promise with functional similar to time.sleep, but non-blocking. Should be used only inside coroutine.

Use like promise:

```
yield Delay(10)  # wait 10 seconds
```

Use like a channel:

```
delay_chan = Delay(10)
while True:
    yield delay_chan.get()
    print('ok!')  # prints 'ok!' every 10 seconds
```

get()
Emulate interface of channels.

process()
Emulate interface of promises. Internal use only!

class microasync.async.**SlidingChannel**(*limit=1*)
Bases: microasync.async.Channel

Channel in which new items overwrites old.

microasync.async.**as_chan**(*create_chan*)
Decorator which creates channel and coroutine. Passes channel as a first value to coroutine and returns that channel.

Usage:

```
@as_chan
def thermo(chan, unit):
    while True:
        yield chan.put(convert(thermo_get(), unit))

@coroutine
def main():
    thermo_chan = thermo('C')
    while True:
        print((yield thermo_chan.get()))  # prints current temperature
```

Parameters **create_chan** (*type[Channel]*) – Type of channel.

Returns Created coroutine.

microasync.async.**clone**(*chan*, *n*, *chan_type=<class 'microasync.async.SlidingChannel'>*)
Creates clones of presented channels.

Usage:

```
chan_1, chan_2 = clone(chan, 2)
```

Parameters

- **chan** (*Channel*) – Original channel.
- **n** (*int*) – Count of clones.
- **chan_type** (*type[U]*) – Type of new channels.

Returns Created channels.

Return type list[U]

microasync.async.**coroutine**(*fnc*)
Decorator for defining coroutine.

Usage:

```
@coroutine
def my_coroutine(x):
    print(x)

my_coroutine()
loop()  # corouitnes starts working only after starting main loop
```

microasync.async.**do_all**(**chans*)
Creates new channel with single item from each of *chans* in sequential order. Should be used only inside coroutine.

Usage:

```
led_state, trig_state = yield do_all(led_chan.get(), trig_chan.get())
```

Parameters **chans** (*Channel*) – Channels from which we need to get items.

Returns Channel in which we put values from *chans*.

Return type Channel

microasync.async.**loop**()
Starts main loop.

microasync.async.**process_all**()
Process all promises. Internal use only!

microasync.async.**select**(**chans*)
Creates a channel with works like fifo for messages from original channels. Works like *select* from go or *alts!* from *core.async*.

Usage:

```
select_chan = select(delay_chan, trigger_chan)
while True:
    chan, val = yield select_chan.get()
    if chan == delay_chan:
        print('delay')
    else:
        print('trig by ', val)
```

Parameters **chans** (*Channel*) – Channels from which we should get items.

Returns Channel with all *chans* items.

Return type Channel

2.1.3 microasync.device module

Experimental non-blocking api for pyboard.

class microasync.device.**FakePyb**
Bases: object

microasync.device.**get_accel**()
Creates channel for on-board accel.

Usage:

```
accel_chan = get_accel()
while True:
    print((yield accel_chan.get()))  # prints current accel (x, y, z)
```

Returns Created channel.

Return type Channel

microasync.device.**get_servo**(*num*)

Creates write and read channels for servo. Should be used only in coroutine.

Usage:

```
servo_set, servo_get = get_servo(1)
yield servo_set.put(90)  # set servo to 90 degrees
print((yield servo.get()))  # prints current servo degree
```

Parameters **num** (*int*) – Number of servo.

Returns Write and read channels.

Return type (Channel, SlidingChannel)

microasync.device.**get_switch**()

Creates channel for onboard switch. Should be used only in coroutine.

Usage:

```
switch = get_switch()
while True:
    yield switch.get()
    print('clicked!')
```

Returns Channel for onboard switch.

Return type Channel

microasync.device.**pyb**

2.1.4 microasync.utils module

Helpers for internal use only!

class microasync.utils.**Atom**(*value=None*)

Bases: microasync.utils.WithEquality

reset(*value*)

swap(*fn*)

class microasync.utils.**Promise**

Bases: microasync.utils.WithEquality

delivery(*value*)

class microasync.utils.**WithEquality**

Bases: object

Class for avoiding micropython limitations with objects equality.

class microasync.utils.**pyb**
Bases: object

classmethod rng()

2.1.5 Module contents

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

m